VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY
DEPARTMENT OF MICROELECTRONICS

TESTOVÁNÍ SRAM PAMĚTÍ S VYUŽITÍM MBIST
SRAM MEMORIES TESTING WITH UTILIZATION OF MEMORY BUILT-IN-SELF-TEST

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan Sedlář**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Edita Hejátková**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Mikroelektronika**
Ústav mikroelektroniky

Student: Bc. Jan Sedlář ***ID:*** 164777
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Testování SRAM pamětí s využitím MBIST

POKYNY PRO VYPRACOVÁNÍ:

Prozkoumejte, zda je možné využít softwarový nástroj Tessent MemoryBIST pro testovaní SRAM pamětí v testovacích čipech společnosti ON Semiconductor. Navrhněte řešení formou vzorového příkladu na předloženém reálném návrhu čipu.
Funkčnost návrhu ověřte simulací. Zhodnoťte dosažené výsledky a možný přínos.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 22.5.2018

Vedoucí práce: Ing. Edita Hejátková
Konzultant: Ing. Jan Zápeca, ON Semiconductor, Brno

doc. Ing. Lukáš Fujcik, Ph.D.
předseda oborové rady

ABSTRAKT

Práce se zabývá testováním pamětí SRAM pomocí metody MBIST s využitím softwarového nástroje Tessent Memory BIST. Cílem je seznámit se s testováním pamětí a vytvořit návrh na specifickém čipu, kterému po implementaci testovací logiky budou zachovány původní vlastnosti a funkce. Následně je provedeno vyhodnocení tohoto nástroje a jeho použitelnost.

KLÍČOVÁ SLOVA

SRAM, Tessent, MBIST, Test paměti, JTAG, Čip

ABSTRACT

The project deals with the testing of SRAM memories using method MBIST with the utilisation of sofware tool Tessent Memory BIST. The main purpose is to get familiar with memory testing and to create a design for testing on a specific chip which after its implementation on the chip will retain the original features and functions. Subsequently, the tool is evaluated on its usability.

KEYWORDS

SRAM, Tessent, MBIST, Memory test, JTAG, Chip

SEDLÁŘ, Jan. *Testování SRAM pamětí s využitím MBIST*. Brno, Rok, 84 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky. Vedoucí práce: Ing. Edita Hejátková

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Testování SRAM pamětí s využitím MBIST“ jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval konzultantovi diplomové práce panu Ing. Janu Zápecovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno ....

podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

Seznam symbolů, veličin a zkratek **9**

Úvod **14**

1 Obecné seznámení s pamětí SRAM **15**
1.1 Zápis/Čtení z paměti 15
1.2 Adresování SRAM 16

2 Testování paměti **17**
2.1 Chybový model paměti 18
2.1.1 Klasifikace chybových primitivů 18
2.1.2 Chyby pro paměti s jedním portem 19

3 Testovací algoritmy **23**
3.1 Algoritmy 23
3.1.1 Porovnání algoritmů 25

4 Seznámení s testovacím čipem **26**
4.1 Moduly s pamětmi 26
4.1.1 ATC 26
4.1.2 MFC 28
4.1.3 SHTC 28
4.1.4 FUNC-MEM 30
4.2 Hierarchie 30

5 BIST **34**
5.1 BIST pro paměti 34
5.2 Nástroj DFT pro paměti 34
5.2.1 Základní MBIST bloky 34
5.2.2 Práce s MBIST 36

6 Návrh DFT pro testovací čip **40**
6.1 Požadavky 40
6.2 Flow 41
6.2.1 Implementace 41
6.2.2 Verifikace 43
6.3 Použití Makefilu 44

7 Implementace DFT **46**
7.1 Limitace nástroje Tessent MBIST 46
7.2 Moduly 47
7.2.1 ATC-DP 47
7.2.2 ATC-SP 50
7.2.3 MFC-DP 50
7.2.4 MFC-SP 56
7.2.5 SHTC-DP 56
7.2.6 SHTC-SP 57
7.2.7 FUNC-MEMORY 58
7.2.8 PADRING 58

8 Výsledky **60**
8.1 Velikost před a po 60
8.2 Logická simulace 61
8.3 STA 61
8.4 Rychlost funkčního testování 66

9 Zásady návrhu pro použití Tessent MBIST **68**

10 Závěr **69**

Literatura **70**

Seznam příloh **71**

A Algoritmy **72**

B Tessent MBIST **74**

C Implementace DFT pro SP paměti **76**

D MBIST časové cesty **83**

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

AC	Alternating Current
ASIC	Application-Specific Integrated Circuit
ATC	Access Time Characterization
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BIST	Built-In Self-Test
CF	Coupling Faul
CFdrd	Deceptive Read Destructive CF
CFds	Disturb CF
CFir	Incorrect Read CF
CFrd	Read Destructive CF
CFst	State CF
CFtr	Transition CF
CFwd	Write Destructive CF
CGC	Clock Gating Cell
CUT	Circuit Under Test
DC	Direct Current
DFT	Design For Testability
DMA	Direct Memory Access
DRAM	Dynamic Random-Access Memory
DRC	Design Rule Checking
DRDF	Deceptive Read Destruction Fault
DRF	Data Retention Fault
FV	Fault Coverage
FFM	Functional Fault Model
FP	Fault Primitiv
GUI	Graphical User Interface
HDL	Hardware Description Language
ICL	Instrument Connectivity Language
IJTAG	Internal JTAG
IP	Intellectual Property
IRF	Incorrect Read Fault
JTAG	Joint Test Action Group
LEC	Logic Equivalence Checking
MBIST	Memory Built-In Self-Test
MFC	Maximum Frequency Characterization
NAF	No Access Faul

PDL	Procedure Description Language
RDF	Read Destruction Faul
RRDF	Random Read Destruction Fault
RRF	Random Read Fault
RTL	Register Transfer Level
SAF	Stuck-At Fault
SF	State Fault
SHTC	Setup/Hold Time Characterization
SIB	Segment Insertion Bit
SOS	Sensitizing Operation Sequence
SRAM	Static Random Access Memory
STA	Static Timing Analysis
SVF	Serial Vector Format
TDR	Test Data Register
TF	Transition Fault
URF	Undefined Read Fault
USF	Undefined State Fault
UWF	Undefined Write Fault
WDF	Write Destruction Fault
WGL	Waveform Generation Language

SEZNAM OBRÁZKŮ

1.1 Bitová buňka jako bistabilní klopný obvod z invertorů 15
1.2 Bitová buňka z šesti tranzistorů 15
1.3 Obecná architektura SRAM[4] 16
2.1 Stromová klasifikace chybového primitivu [2] 18
4.1 ATC s dlouhým přístupovým časem 26
4.2 ATC s krátkým přístupovým časem 27
4.3 ATC blokové schéma pro charakterizaci přístupového času paměti . . 27
4.4 Legenda pro bloková schémata modulů 27
4.5 MFC blokové schéma pro charakterizaci maximální operační frekvence 28
4.6 SHTC setup/hold 29
4.7 SHTC blokové schéma pro charakterizaci setup a hold času 29
4.8 SHTC-DP blokové schéma 30
4.10 Rozložení na čipu 32
4.9 FUNC-MEM blokové schéma 33
5.1 Ukázka blokově vygenerované testovací logiky MBIST a JTAG 35
5.2 Postup vložení testovací logiky pomocí Tessent MBIST 38
6.1 Návrh DFT topologie 41
6.2 Flow 42
6.3 MAKEFILE prostředí 45
7.1 Napojení dvou BYPASS signálů 46
7.2 ATC-DP blokové schéma po vložení testovací logiky paměti 48
7.3 ATC-DP připojovací modul 49
7.4 MFC-DP blokové schéma po vložení testovací logika paměti 51
7.5 MFC-DP požadovaná testovací logiky paměti 52
7.6 MFC-DP připojovací modul 53
7.7 MFC-DP nahrazený MBIST 54
7.8 MFC-DP postup nastavení oscilátoru přes IJTAG síť 55
7.9 SHTC-DP blokové schéma po vložení testovací logiky paměti 57
7.10 FUNC-MEM blokové schéma po vložení testovací logiky paměti . . . 59
8.1 Výsledná DFT 60
8.2 Kritické časové cesty pro modul ATC 62
8.3 Kritické časové cesty pro modul SHTC 63
8.4 Kritické časové cesty pro orig. modul MFC 65
8.5 Kritické časové cesty pro generovaný modul MFC 65
B.1 Ukázka MBIST rozhraní pro paměť s 4-bitovou sběrnicí 74
B.2 Šablona paměti pro MBIST 75

C.1 ATC-SP blokové schéma pro charakterizaci přístupového času jednoportové paměti 76
C.2 ATC-SP blokové schéma po vložení testovací logiky paměti 77
C.3 ATC-SP připojovací modul 78
C.4 MFC-SP blokové schéma 79
C.5 MFC-SP blokové schéma po vložení testovací logiky paměti 79
C.6 MFC-SP požadovaná testovací logika paměti 80
C.7 MFC-SP připojovací modul 81
C.8 SHTC-SP blokové schéma po vložení testovací logiky paměti 82
D.1 Nejdelší časová cesta pro původní MBIST 83
D.2 Nejdelší časová cesta pro generovaný MBIST 84

SEZNAM TABULEK

2.1 Seznam chyb pro 1PF1 FFMs 20
3.1 Časová náročnost algoritmu na jeho složitosti a velikosti paměti . . . 23
3.2 Instrukce pro testovací algoritmy 23
3.3 Jednotlivé algoritmy 25
3.4 Pokrytí chyb pro jednu buňku 25
3.5 Pokrytí chyb pro dvě buňky 25
8.1 Nárůst plochy designu 61
8.2 Změny v kritických cestách pro paměti modulu ATC-SP 62
8.3 Změny v kritických cestách pro paměti modulu ATC-DP 63
8.4 Změny v kritických cestách modulu SHTC-SP a SHTC-DP 64
8.5 Časy v kritických cestách modulu MFC-DP 66
8.6 Časy v kritických cestách modulu MFC-SP 66
A.1 Jednotlivé algoritmy 72
A.2 Pokrytí chyb pro jednu buňku 73
A.3 Pokrytí chyb pro dvě buňky 73

ÚVOD

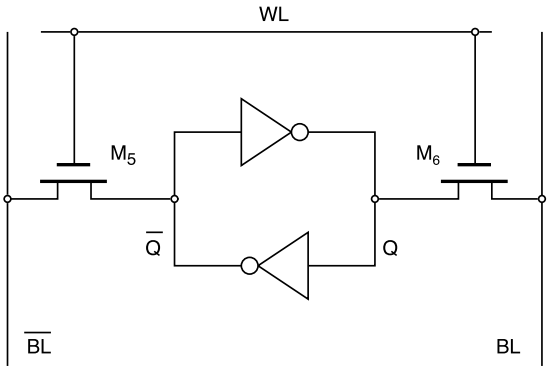
V době, kdy jsou kladeny na návrháře požadavky na co nejmenší časové rozestupy mezi návrhem a výrobou, je nutné minimalizovat čas strávený při návrhu řešení testovatelnosti a jeho rychlosti.

Toto zrychlení na poli návrhu čipů přináší nástroje, které dokážou automaticky vytvářet testovací logiku na běžné prvky jako paměti. Takovým je Tessent MBIST. Tento nástroj bude vyzkoušen pro vytváření testovací struktury pro paměti v testovacím čipu. Jelikož je také charakterizuje, nelze jej vždy použít standardním způsobem. Potenciální překážky způsobené nestandardním použitím nástroje budou zaznamenány a způsoby jejich řešení rozepsány a vyzkoušeny. Z poznatků, které se budou týkat pouze návrhu front-endu, nikoli back-endu, bude vytvořeno doporučení o použitelnosti nástroje v budoucích projektech.

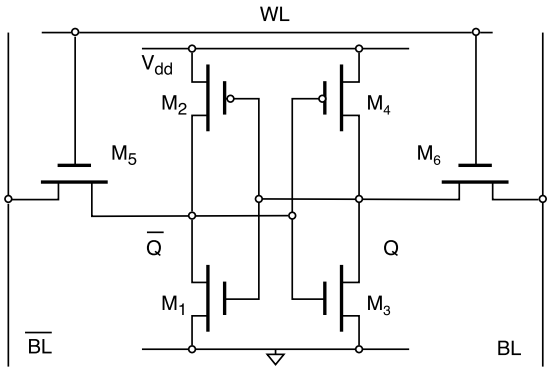
Práce se skládá ze dvou částí: teoretické a praktické. Teoretická část je rozdělena na seznámení se SRAM pamětí, jejími chybami a rozbor testovacích algoritmů. Část praktická se potom dělí na seznámení s testovacím čipem a MBISTem, požadavky na design, flow, implementace a výsledky.

1 OBECNÉ SEZNÁMENÍ S PAMĚTÍ SRAM

Static Random Access Memory (SRAM) je druh polovodičové paměti, která se vyznačuje vysokou rychlostí, větší složitostí a s tím spojenou i větší plochou. Jelikož se jedná o paměť statickou, nemusí být v pravidelných intervalech obnovována jako u paměti dynamické (DRAM). Tento typ paměti si udrží uložené hodnoty až do odpojení napájení.



Obr. 1.1: Bitová buňka jako bistabilní klopný obvod z invertorů



Obr. 1.2: Bitová buňka z šesti tranzistorů

Bitová buňka je sestavena ze dvou invertorů jako bistabilní klopný obvod. Pro práci s ní musíme aktivovat řádkový vodič WL, který je napojen na tranzistory $M5$ a $M6$. Tyto tranzistory oddělují paměťovou buňku od datových vodičů BL a $\overline{BL}$. Bez aktivace WL vodiče by operace čtení a zápis nemohly proběhnout.

SRAM paměti je možné vytvořit několika způsoby, například jako 6 tranzistorové z obr. 1.2 nebo 4 tranzistorové s dvěma rezistory.

1.1 Zápis/Čtení z paměti

Na 6-ti tranzistorové buňce SRAM z obr.1.2 bude názorně ukázáno, jakým způsobem se z buňky čte a zapisuje.

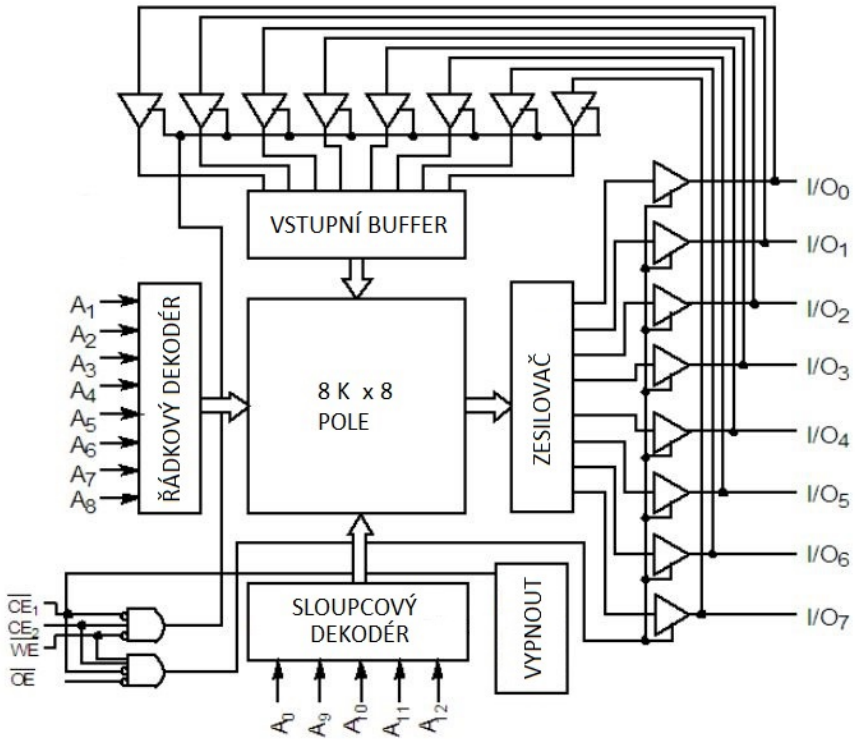
Zápis nového stavu do buňky začíná zapsáním zvolené logické hodnoty do datových vodičů BL a $\overline{BL}$. Poté je přiveden signál na WL. Ten otevře tranzistory ($M5$ a $M6$) a datové vodiče BL a $\overline{BL}$ se připojí na buňku, kterou překlopí do požadovaného stavu. BL a $\overline{BL}$ vodiče s tranzistory ($M5$ a $M6$) jsou silnější než tranzistory, které tvoří paměťovou buňku ($M1$, $M2$, $M3$, $M4$). Je to z toho důvodu, aby je přemohly a dokázaly přepsat.

Je-li potřeba hodnotu z buňky přečíst, stačí otevřít tranzistory ($M5$ a $M6$) pomocí vodiče WL a přečíst ji z datového BL vodiče, ale kvůli parazitní kapacitě

datových vodičů a tlaku na zvýšení rychlosti, je uplatněn složitější postup. Nejprve jsou datové vodiče BL a $\overline{BL}$ nabity na napěťové hodnoty mezi jedničkou a nulou. Po nabití je nastaven WL vodič, který otevře tranzistory ($M5$ a $M6$), následně se na datových vodičích jemně změní napětí, které představuje stav buňky. Napětí je zachyceno diferenciálním zesilovačem z vodiče BL a $\overline{BL}$. Zesilovač následně vyhodnotí, v jakém stavu se buňka nachází. Čím menší napětí je požadováno na vstupu zesilovače, tím je operace čtení rychlejší. Naopak, čím větší rozdílové napětí je na vodičích požadováno k rozeznání stavu buňky, tím je doba nabíjení kapacity vodiče na požadovanou úroveň delší, a tím je i delší doba čtení.

1.2 Adresování SRAM

Adresování jednotlivých buněk je rozděleno na sloupce a řádky. Příklad na obr. 1.3 představuje paměť s 8-bitovým slovem a hloubkou 8k, která má 13-bitovou adresu, kdy bity dva až osm rozhodují řadu a zbylých pět rozhoduje sloupec. Fyzické rozmístění buněk se nemusí shodovat s adresou na vstupu. Tato informace není nutná při užívání paměti, ale je důležitá při testování. Při testování je nutné znát, jak se fyzické buňky mapují na adresu, aby se instrukce vykonaly na správných buňkách a nedošlo tak k chybě.



Obr. 1.3: Obecná architektura SRAM[4]

2 TESTOVÁNÍ PAMĚTI

Testovat paměti je důležité ve všech fázích jejího života, od výroby až po běžné používání. Ve výrobě se funkční testy používají pro rozhodování, zda je paměť v pořádku či ne. Při používání se mohou provádět zjednodušené testy před spuštěním, které můžou sloužit jako varování před selhávající pamětí. Tyto testy bývají často implementovány do paměti v podobě Memory Built-In Self-Test (MBIST), se kterou je možné testovat paměť bez speciálního zařízení. Paměti se testují na nejrůznější funkční chyby, které jsou popsány o kapitolu níže. Při charakterizaci pamětí je běžné testovat jejich vlastnosti jak ze stránky elektrické, tak ze stránky časové, aby bylo zajištěno, zda parametry paměti odpovídají dané specifikaci. Přehled možných testů lze shlédnout níže.

- DC
  - Kontaktní test
  - Charakteristiky napájení
  - Svodové proudy
  - Prahový test
  - Test budiče proudového výstupu
  - Zkouška zkratového proudu výstupu
- AC
  - Doba náběhu/sestupu pulzu
  - Setup/Hold čas
  - Přístupová doba
  - Test maximální frekvence
- Funkční test.
  - Chyby buněk
  - Chyby adresního dekodéru
  - Chyby zapisovací/čtecí logiky

Poruchy pamětí můžou vznikat při návrhu, výrobě a dále vlivem času a prostředí. Možné příčiny jsou:

- Špatný návrh
- Koroze
- Iontová kontaminace (způsobí změnu prahového napětí tranzistoru)
- Elektromigrace (zmenšování vodičů na čipu -> nárůst rezistivity)
- Radiace a kosmické záření

2.1 Chybový model paměti

Abychom se mohli se bavit o chybách pamětí a jejich nalezení, je nutné zavést funkční chybový model, který popisuje způsob, jakým selhaly. Provedením několika paměťových operací při sledování chování funkční části paměti lze chyby definovat jako odlišné chování od toho specifikovaného. Pro popis jakéhokoliv chybového modelu jsou proto podstatné dvě věci.

- Seznam provedených operací na paměti
- Seznam odpovídajících odchylek od očekávaného chování

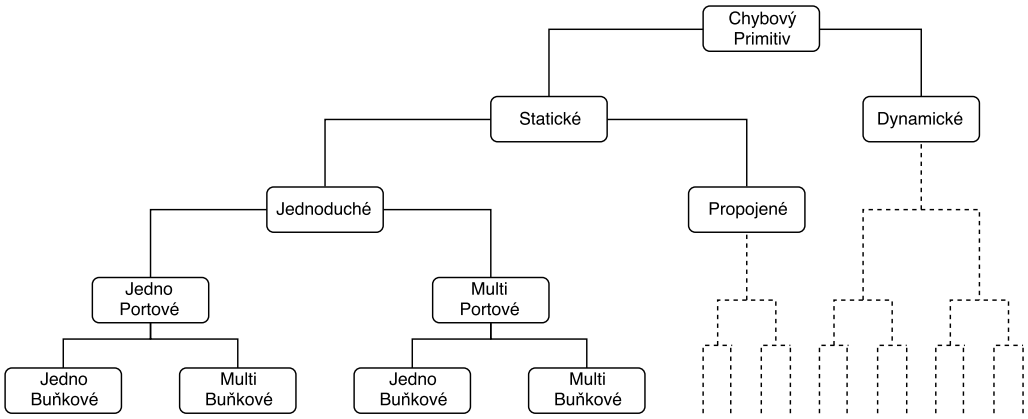
Jakýkoliv seznam provedených operací na paměti se, nazývá operační sekvence. Operační sekvence, která vyústí v rozdíl mezi pozorovaným a očekávaným chováním paměti se nazývá Sensitizing Operation Sequence (SOS).

Pro specifikování jedné chyby je nutné specifikovat SOS spolu s odpovídajícím chybovým chováním. Tato kombinace se nazývá Fault Primitiv (FP) a je zapisována v podobě <S/F/R>. Jako první je uveden aktivační SOS pod písmenem S, který danou chybu aktivuje. Druhé písmeno F značí chybovou hodnotu paměti (například po zápisu log. 1 není v paměti log. 1, ale 0). Třetí písmeno R popisuje logickou hodnotu po operaci čtení na výstupu paměti.

Koncept FP nám umožňuje vytvořit kompletní soustavu všech teoretických chyb za předpokladu, že známe všechny povolené operační sekvence pro danou paměť.

2.1.1 Klasifikace chybových primitivů

Chyby jsou děleny do několika kategorií znázorněných na obr. 2.1.



Obr. 2.1: Stromová klasifikace chybového primitivu [2]

Statické a dynamické chyby

Pro aktivování statických chyb je nanejvýše potřeba jedna sekvenční operace. Pro

aktivování dynamické chyby je potřeba více než jedna sekvenční operace. Tato chyba může postihovat jakoukoliv část paměti.

Jednoduché a propojené chyby

Jednoduché chyby se projevují jako chyby, které nemohou ovlivnit chování ostatních buněk. To znamená, že chování jednoduché chyby nemůže změnit chování jiné. Propojené chyby se projevují jako chyby, které ovlivňují chování ostatních buněk. To znamená, že chování určité chyby může způsobit změnu v chování jiné a může docházet k maskování.

Jednoportové a víceportové chyby

Jednoportové chyby (1PFs) potřebují maximálně jeden port k tomu, aby aktivovaly chyby. Víceportové chyby (pPfs) se aktivují pouze při provedeni dvou nebo více současně probíhajících operací na rozdílných portech.

Například, pokud jediná čtecí operace z jednoho portu použitá na buňku $C1$ způsobí té samé buňce změnu hodnoty, můžeme to nazvat jednoportovou chybou (1PFS). Pokud dvě současné čtecí operace ze dvou portů způsobí změnu hodnoty dané buňky, můžeme ji nazvat jako dvouportovou chybu (2PFS). Stejným způsobem můžeme pokračovat v charakterizaci dalších portových chyb.

Jednobuňkové a vícebuňkové chyby

Nechme C definovat jako počet různých buněk, ke kterým bylo přistoupeno během SOS operace. Například, pokud se operační sekvence skládá pouze z jedné čtecí operace, použité pouze na jednu buňku, můžeme říct, že $C = 1$. Pokud se operační sekvence skládá ze dvou čtecích operací po sobě jdoucích na rozdílné buňky, tak $C = 2$.

Jednobuňkové chyby se týkají jen FP s jednou buňkou. Buňka, která je aktivována, je ta samá, na které se chyba objeví. Spojené chyby se vyskytují při FP operacích, kdy je zapojena do testu více než jedna buňka. Tyto buňky mají schopnost aktivovat chybu nebo k ní přispět v jiné buňce. Tato chyba není omezena pouze na jednu agresivní buňku, ale těch, které ovlivňují obětní buňku, může být i více.

2.1.2 Chyby pro paměti s jedním portem

Tato část je zaměřena na seznámení se s chybami paměti obsahující jeden port. Chyby pamětí s jedním portem můžeme rozdělit na chyby, které vznikají a postihují pouze jednu paměťovou buňku (1PF1) a na chyby, které vznikají na základě více paměťových buněk (1PFx). Bude nutné se zabývat chybami do dvou buněk, protože většina chyb se nachází v tomto spektru.

Chyby, které postihují pouze jednu buňku (1PF1), mají takovou vlastnost, že po aplikaci aktivační sekvence chyby na danou buňku se chyba objeví pouze v dané buňce.

Chyby závislé na dvou buňkách (1PF2), kdy jedna buňka je obětní (cv) a druhá buňka je agresor (ca), můžeme rozdělit do tří kategorií.

- 1PF2s, pokud je buňku agresora v určitém stavu, aktivuje se chyba v buňce obětní
- 1PF2, pokud je na buňku agresora aplikována určitá operace (čtení nebo zápis), dojde k aktivování chyby v buňce oběti
- 1PF2v, pokud je buňka agresora v určitém stavu a je na obětní buňku aplikována operace (čtení nebo zápis), dojde k aktivovaní chyby v této buňce

Chybové primitivy budeme zapisovat pro každou chybu takto $<S/F/R>$.

Znak S bude popisovat aktivační sekvenci chyby. Například S může nabývat sekvencí (1, 0, 0w1, 1w1, 1r1 a další), kde první 1 nebo 0 znamená původní hodnotu paměti a následná operace w0/1 nebo r0/1 znamená zápis či čtení dané hodnoty. Operace 1r1 vyjadřuje operaci čtení z buňky obsahující hodnotu 1 a ta samá hodnota je očekávána i na výstupu po provedení této operace.

Znak F bude popisovat chybovou hodnotu buňky po aktivaci chyby. Například F může nabývat hodnot (0, 1 a ?). Pokud je buňka po aktivaci chyby ve stavu "?", může se nacházet v jedničce nebo nule.

Znak R bude popisovat logickou hodnotu na výstupu SRAM, pokud je na buňku aplikována čtecí operace, která aktivuje chybu. Například R může nabývat hodnot (0, 1, ? a -), kdy "-" znamená, že žádná hodnota není na výstupu, protože aktivační sekvence neobsahuje čtecí operaci.

Tab. 2.1: Seznam chyb pro 1PF1 FFMs

#	**FFM**	**Chybový primitiv**
1	SF	<1/0/->, <0/1/->
2	TF	<0w1/0/->, <1w0/1/->
3	WDF	<0w0/1/- >, <1w1/0/- >
4	RDF	<0r0/1/1 >, <1r1/0/0 >
5	DRDF	<0r0/1/0 >, <1r1/0/1 >
6	RRDF	<0r0/1/? >, <1r1/0/? >
7	IRF	<0r0/0/1 >, <1r1/1/0 >
8	RRF	<0r0/0j? >, <1r1/1/? >
9	USF	<1/?/->, <0/?/->
10	UWF	<0w0/?/- >, <0w1/?/- >, <1w0/?/- >, <1w1/?/- >
11	URF	<rx/? /0 >, <rx/? /1 >, <rx/? /? >
12	SAF	<V/0/->, <V/1/->

Tab. 2.1 Pokračuje

Pokračování Tab. 2.1

#	FFM	Chybový primitiv
13	NAF	{<0w1/0/->, <1w0/1/->, <rx/x/? >}
14	DRF	<lT/0/->, <0T/1/->, <xT/?/->

1. **State Fault (SF)** - buňka má původní logickou hodnotu převrácenou před tím, než je k ní možno přistoupit. Tato chyba nezávisí na žádné operaci, ale na původní logické hodnotě. SF má dvě aktivační chybové sekvence <0/1/-> a <1/0/->.
2. **Transition Fault (TF)** - buňka je postihnuta touto chybou, pokud se nepovede přechod, a to buď z 0 -> 1 nebo 1 -> 0. Tato chyba je aktivována pomocí zapisovací operace hodnoty opačné původního stavu. TF má dvě aktivační chybové sekvence <0w1/0/-> a <1w0/1/->.
3. **Write Destruction Fault (WDF)** - buňka je postihnuta touto chybou, pokud provedeme operaci zápisu hodnoty shodné v buňce, ale logická hodnota v buňce se následně překlopí. WDF má dvě aktivační chybové sekvence <0w0/1/-> a <1w1/0>.
4. **Read Destruction Fault (RDF)** - buňka je postihnuta touto chybou, pokud provedeme čtecí operaci, ale buňka překlopí svou logickou hodnotu a také ji vrátí na výstupu. RDF má dvě aktivační chybové sekvence <0r0/1/1> a <1r1/0/0>.
5. **Deceptive Read Destruction Fault (DRDF)** - buňka je postihnuta touto chybou, pokud provedeme čtecí operaci a buňka překlopí svou logickou hodnotu, ale vrátí svoji původní hodnotu. DRDF má dvě aktivační chybové sekvence <0r0/1/0> a <1r1/0/1>.
6. **Random Read Destruction Fault (RRDF)** - buňka je postihnuta touto chybou, pokud provedeme čtecí operaci a buňka překlopí svou logickou hodnotu, ale vrátí náhodnou hodnotu. RRDF má dvě aktivační chybové sekvence <0r0/1/?> a <1r1/0/?>.
7. **Incorrect Read Fault (IRF)** - buňka je postihnuta touto chybou, pokud provedeme čtecí operaci, ale logická hodnota na výstupu je špatná s tím, že uložená hodnota v buňce se nezměnila. IRF má dvě aktivační chybové sekvence <0r0/0/1> a <1r1/1/0>.
8. **Random Read Fault (RRF)** - buňka je postihnuta touto chybou, pokud provedeme čtecí operaci, ale logická hodnota na výstupu je náhodná s tím, že uložená hodnota v buňce se nezměnila. RRF má dvě aktivační chybové sekvence <0r0/0/?> a <1r1/1/?>.
9. **Undefined State Fault (USF)** - buňka, která je postihnuta tuto chybou má původní logickou hodnotu změněnou na náhodnou hodnotu, než je k ní

možno přistoupit. Tato chyba nezávisí na žádné operaci, ale na původní logické hodnotě. USF má dvě aktivační chybové sekvence <0/?/-> a <1/?/->.

10. **Undefined Write Fault (UWF)** - buňka je postihnuta touto chybou, pokud provedeme zápisovou operaci a hodnota buňky se změní na náhodnou hodnotu. UWF má čtyři aktivační chybové sekvence <0w0/0->, <0w1/?/->, <1w0/?/-> a <1w1/?->.
11. **Undefined Read Fault (URF)** - buňka je postihnuta touto chybou, pokud provedeme operaci čtení a tato operace změní hodnotu buňky do náhodného stavu, výstupní hodnota paměti může být správná, špatná, ale i náhodná. URF má šest aktivačních chybových sekvencí <0r0/?/0>, <0r0/?/1>, <0r0/?/?>, <1r1/?/0>, <1r1/?/1> a <1r1/?/?>.
12. **Stuck-At Fault (SAF)** - buňka je postihnuta touto chybou, pokud provedeme jakoukoliv operaci čtení či zápisu a hodnota buňky se nezmění. SAF má dvě aktivační chybové sekvence <V/0/-> a <V/0/->. Kde "V"je jakákoliv aktivační operace.
13. **No Access Fault (NAF)** - buňka není přístupná, není ji možné změnit zápisem a není z ní možné číst, při operaci čtení získáme na výstupu pouze náhodné hodnoty. NAF má 4 aktivační chybové sekvence <0w1/0/->, <1w0/1>, <0r0/1/?> a <1r1/0/?>.
14. **Data Retention Fault (DRF)** - buňka je postihnuta touto chybou, pokud počkáme dobu delší, než je doba přednabíjecího cyklu v SRAM paměti. Pokud nepočkáme tuto dobu a zjistíme chybu, tak se nejedná o chybu DRF, ale State Fault (SF). Pokud se hodnota změnila po delší době, je to chyba DRF. DRF má 4 aktivační chybové sekvence <1T/0/->, <0T/1>, <1T/?/-> a <0T/?/->.

3 TESTOVACÍ ALGORITMY

V předchozí kapitole byly vyjmenovány všechny statické teoretické chyby jedné buňky pro jednoportovou paměť. K tomu, aby byly otestovány, musí existovat algoritmus, který otestuje co nejvíce chyb za co nejkratší dobu. Algoritmy navržené na takový úkol ale nesmí překročit kvadratickou mez přístupů pro jednu buňku, jinak jsou testovací časy nereálné, viz 3.1. Jelikož reálná paměť určité chyby vůbec mít nemusí, je dobré znát design paměti a podle toho vybírat chyby na testování. Se zvolenými chybami k otestování a podle časových požadavků je vybrán vhodný testovací algoritmus. Zvolený algoritmus se může nadále modifikovat pro zachycení specifických chyb nalezených pro danou paměť.

Tab. 3.1: Časová náročnost algoritmu na jeho složitosti a velikosti paměti

Velikost	**Testovací perioda 10ns**			
n	$64n$	$nlog_2n$	$n^{3/2}$	n^2
16k	0.01 s	0.0023 s	0.021 s	2.7 s
64k	0.04 s	0.01 s	0.167 s	42 s
256k	0.17 s	0.047 s	1.34 s	11.4 min.
1M	0.67 s	0.21 s	10.7 s	183 min.
4M	2.68 s	0.92 s	85.9 s	49.2 hod.
16M	10.8 s	4.03 s	11.4 min.	36.5 dne
64M	43.2 s	16.2 s	91.6 min.	584 dnů

3.1 Algoritmy

V této části jsou představeny čtyři základní algoritmy a následně je provedeno srovnání různých algoritmů s jejich pokrytím chyb. K porozumění algoritmů je nutné znát možné manipulace s pamětí viz tab. 3.2.

Tab. 3.2: Instrukce pro testovací algoritmy

#	**Příkaz**	**Popis**
1	R0, R1	Přečti buňku a očekávej určenou hodnotu
2	W1, W0	Zapiš do buňky určenou hodnotu
3	⇳ (...);	Vykonej příkazy v závorce v jakémkoliv adresovacím pořadí
4	⇑ (...);	Vykonej příkazy v závorce od nejnižší adresy po nejvyšší
5	⇓ (...);	Vykonej příkazy v závorce od nejvyšší adresy po nejnižší

Algoritmus 0-1

Jedná se o jeden z nejjednodušších testovacích algoritmů a má následující sekvenci $\Updownarrow (W0); \Updownarrow (R0); \Updownarrow (W1); \Updownarrow (R1)$. Na každou buňku jsou aplikovány pouze čtyři příkazy, jinak řečeno 4N, kdy N značí počet buněk a pronásobením získáme celkový počet operací pro tento test. Pokud by paměť trpěla pouze chybami typu SAF a TF, tak by tento test otestoval paměť na 100 %. Jelikož reálná paměť však může obsahovat více typů chyb, je využití tohoto algoritmu omezené.

Walking

Walking algoritmus má vždy jednu buňku (adresu), která je v odlišném stavu než všechny ostatní buňky v paměti. Tato adresa se po každém cyklu posune, od toho se odvíjí také název walking. Na začátku testu musí být celá paměť nastavena do nul nebo jedniček. Sekvence algoritmu je $\Uparrow (R0, W1, R1, W0)$. Algoritmus dokáže pokrýt chyby typu SF, TF, RDF, IRF a polovinu dvoubuňkových chyb. Je však časově velmi náročný.

Marching

Marching algoritmus změní stav buňky na dané adrese a posune se o adresu dál bez vrácení buňky do původního stavu. Když bude tento algoritmus v polovině, v jedné půlce paměti budou nové hodnoty a v druhé hodnoty staré. Na konci bude mít celá paměť nové hodnoty. Sekvence tohoto algoritmu je $\Uparrow (R0, W1, R1)$. Tento algoritmus má mnoho modifikací a jejich schopnost pokrýt určité chyby se mění, viz tab.3.4.

Galloping

Jde o algoritmus, který v jeho nezměněné podobě je časově velmi náročný se složitostí n^2 a dokáže pokrýt více chyb než algoritmus walking. Sekvence tohoto algoritmu je následující.

1. Do celé paměti je zapsáno 1 nebo 0.
2. Na aktuální adresu je zapsána nová hodnota.
3. Přečtou se všechny ostatní buňky, zda mají svou předpokládanou hodnotu.
4. Adresa je posunuta na novou pozici.
5. Pokračuje se bodem 2 dokud se nedojde na konec paměti.

Pokrytí chyb tímto nezměněným algoritmem je velice vysoký, ale pro svou dobu trvání nepoužitelný, maximálně pro fázi vývoji paměti, kdy se paměť může testovat déle.

3.1.1 Porovnání algoritmů

V dnešní době existuje velké množství algoritmů a je na návrháři, aby vybral ten správný. V tab. 3.3 máme názornou sadu 14 testů s jejich postupy. Každý algoritmus má jinou délku, viz tabulka 3.4, a také se liší tím, jaké chyby dokáže zachytit, což je možné zjistit z tabulek 3.4 a 3.5. Celý výpis algoritmů se nachází v příloze A.

Tab. 3.3: Jednotlivé algoritmy

#	**Test**	**Algoritmus**	**Zdroj**
8	Galpat	$\{\Uparrow (W0); \Uparrow b(W1b, \Uparrow -b(R0, R1b); W0b;)$ $\Uparrow (W1); \Uparrow (W0b, \Uparrow -b(R0, R1b);, W1b);\}$	[2]
9	Walking	$\{\Uparrow (W0); \Uparrow b(W1b, \Uparrow b(R0); R1b; W0b;)$ $\Uparrow (W1); \Uparrow (W0b, \Uparrow b(R0); R1b, W1b);\}$	[2]
14	March SS	$\{\Updownarrow (W0); \Uparrow (R0, R0, W0, R0, W1);$ $\Uparrow (R1, R1, W1, R1, W0); \Downarrow (R0, R0, W0, R0, W1);$ $\Downarrow (R1, R1, W1, R1, W0); \Updownarrow (R0);\}$	[5]

Tab. 3.4: Pokrytí chyb pro jednu buňku

#	**Test**	**Délka**	**SF**	**TF**	**WDF**	**RDF**	**DRDF**	**IRF**	**FC**
8	Galpat	6n+4nRC	2/2	2/2	0/2	2/2	2/2	2/2	10/12
9	Walking	8n+2nRC	2/2	2/2	0/2	2/2	0/2	2/2	8/12
14	March SS	22n	2/2	2/2	2/2	2/2	2/2	2/2	12/12

Tab. 3.5: Pokrytí chyb pro dvě buňky

Test	**CFst**	**CFds**			**CFtr**	**CFwd**	**CFrd**	**CFdr**	**CFir**	**FC**
		xrx	xr-x	xwx						
Galpat	8/8	8/8	4/8	0/8	4/8	0/8	8/8	4/8	8/8	44/72
Walking	8/8	8/8	4/8	0/8	4/8	0/8	8/8	0/8	8/8	40/72
March SS	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	72/72

4 SEZNÁMENÍ S TESTOVACÍM ČIPEM

Čip byl navržen pro testování a charakterizaci SRAM pamětí za účelem ověření funkčnosti a dosažení požadovaných parametrů zákazníkem. Disponuje proto funkcemi pro získání parametrů paměti jako rychlost, časování a spotřeba. Základní funkcí každé paměti je přímý přístup k paměti (DMA).

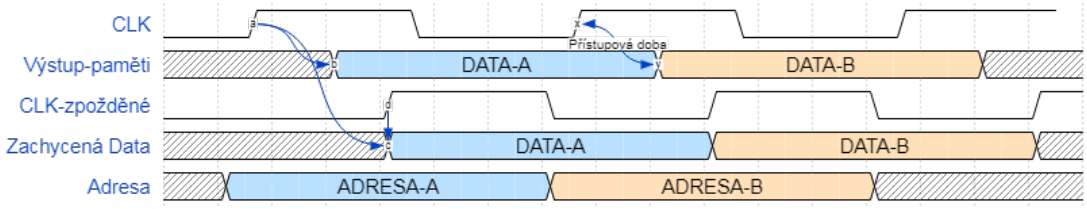
DMA přístup je navržen tak, aby byla vždy aktivní jen jedna paměť. Tato metoda dovoluje použít jednu vstupní a výstupní sběrnici pro všechny paměti. Tímto způsobem čip nepotřebuje samostatné piny pro každou paměť. Přístup k jednotlivým pamětem se nastavuje pomocí registrů v adresovatelné paměti na čipu.

4.1 Moduly s pamětmi

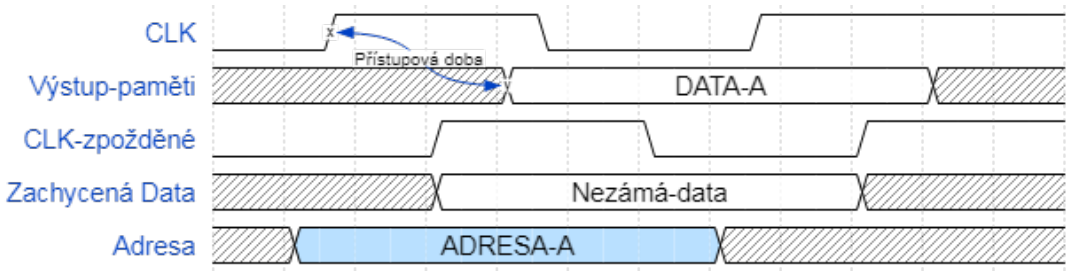
Paměti jsou v čipu rozděleny do jednotlivých modulů, které provádí jednu specifickou charakterizační funkci. Všechna bloková schémata modulů se budou řídit od této části podle legendy na obr. 4.4.

4.1.1 ATC

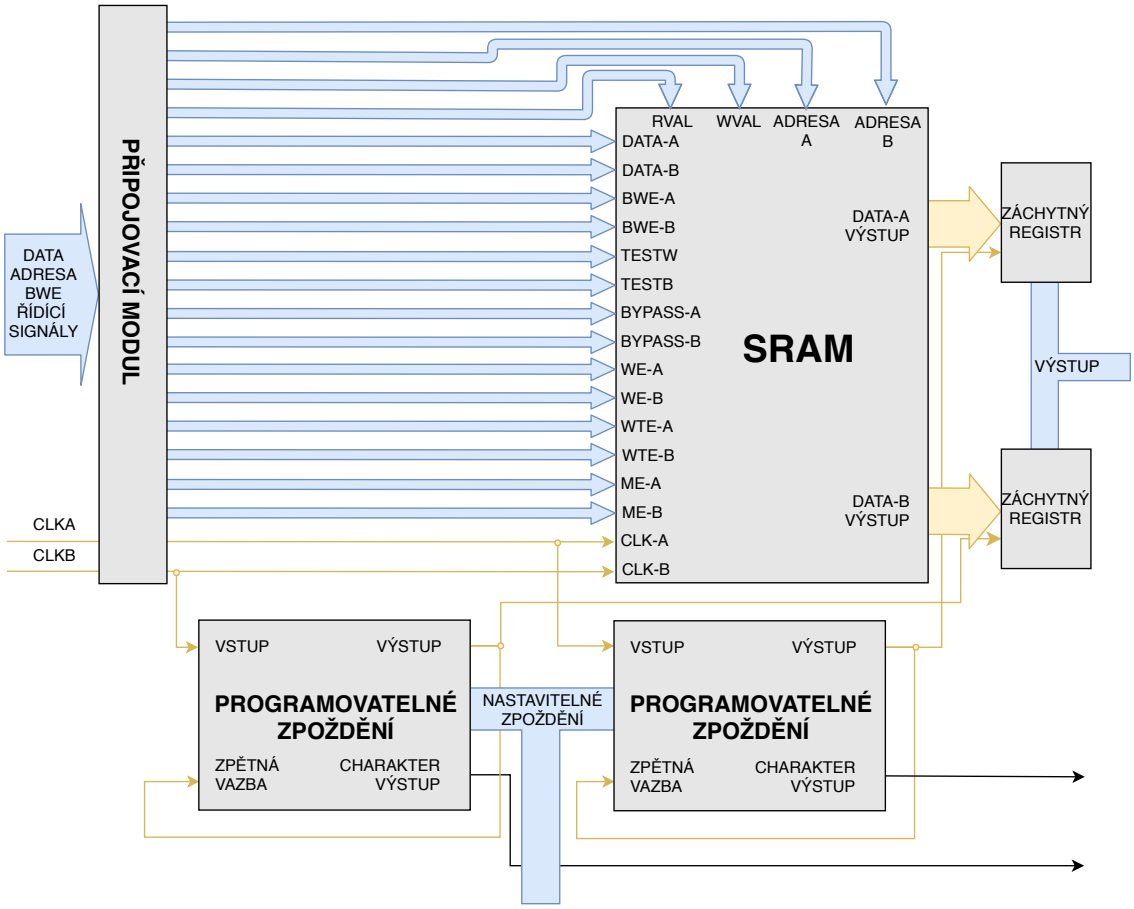
Modul charakterizace přístupového času ATC má zjistit čas potřebný na propagaci dat z paměti na výstup po náběžné hraně hodinového signálu. Jeho blokové schéma je znázorněno na obr. 4.3. Přístupový čas je měřen pomocí záchytných registrů, které jsou napojeny na výstup z paměti. Záchytný registr má proto v cestě hodinového signálu nastavitelné zpoždění. Pokud se nastaví nulové zpoždění, náběžná hrana hodinového signálu dojde současně na paměť a záchytný registr. V takovém případě se nestihnou zpropagovat data z paměti a na záchytném registru nebudou validní data, viz obr.4.2. Nastavitelné zpoždění je zvyšováno, dokud se na záchytném registru neobjeví validní data z paměti, viz obr.4.1. Přístupový čas získáme z nastaveného zpoždění tak, že přepne z této funkce na oscilátor a jeho pulsy budou spočítány a převedeny na frekvenci. Poloviční perioda této frekvence je přístupový čas.



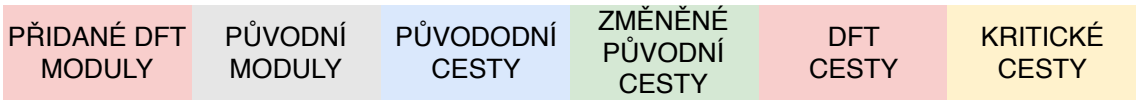
Obr. 4.1: ATC s dlouhým přístupovým časem



Obr. 4.2: ATC s krátkým přístupovým časem



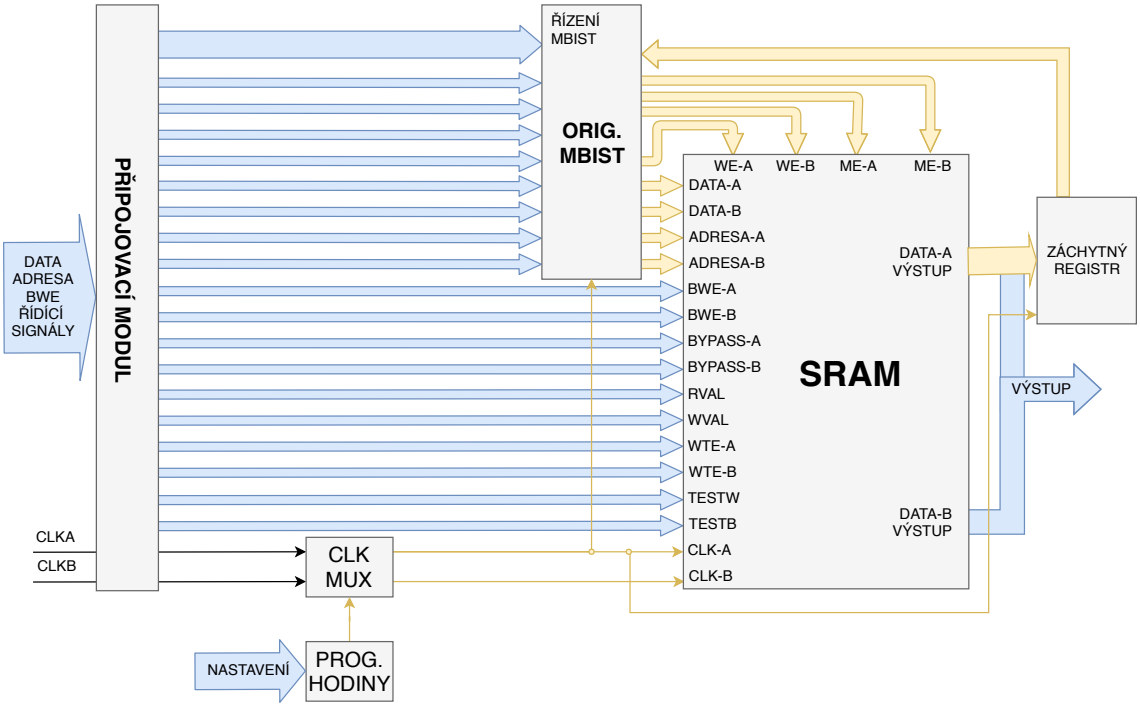
Obr. 4.3: ATC blokové schéma pro charakterizaci přístupového času paměti



Obr. 4.4: Legenda pro bloková schémata modulů

4.1.2 MFC

Modul charakterizace maximální frekvence MFC má charakterizovat maximální frekvenci, na které paměť funguje. Blokové schéma je znázorněno na obr. 4.5. Charakterizace je to dosaženo pomocí MBISTu, který testuje paměť na různých frekvencích. Frekvence je generována nastavitelným oscilátorem, který řídí hodinové porty paměti a MBISTu. Oscilátor je schopný generovat frekvence násobně větší než teoretická frekvence paměti. Postupnou iterací nastavení oscilátoru a testování paměti za použití binárního vyhledávání získáme nastavení oscilátoru reprezentující max. frekvenci. Výsledná maximální frekvence se získá napojením výstupu nastaveného oscilátoru do čítače po určitou dobu a počet sčítaných pulzů se převede na frekvenci.

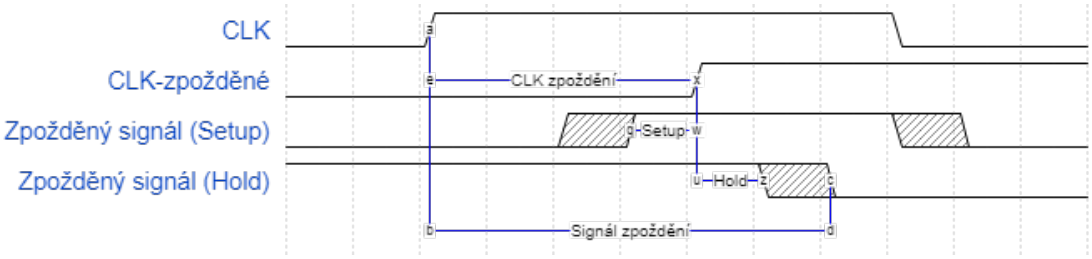


Obr. 4.5: MFC blokové schéma pro charakterizaci maximální operační frekvence

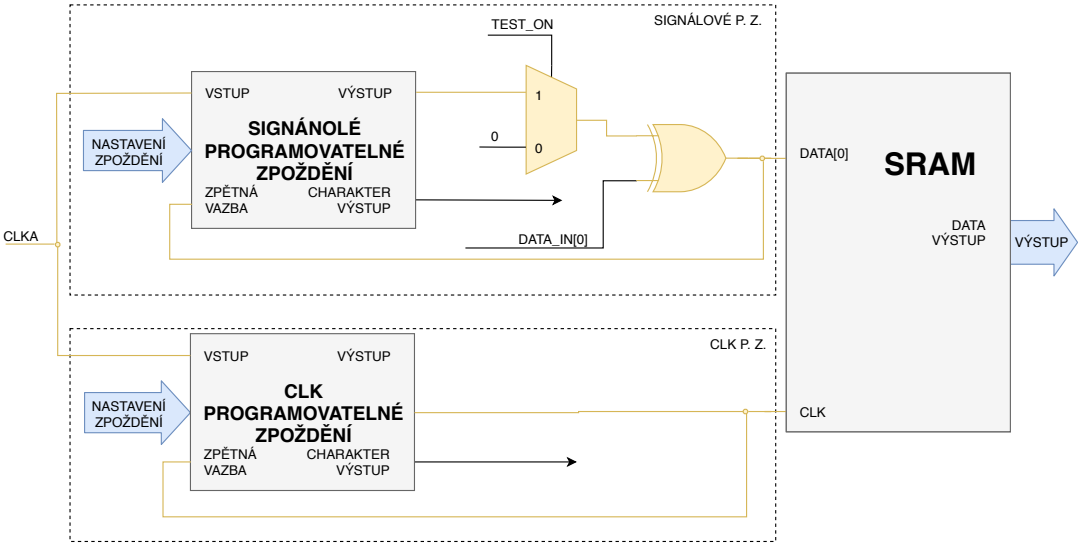
4.1.3 SHTC

Modul charakterizace setup/hold času SHTC má zjistit setup a hold časy pro porty A, D, BWE, ME a WE. Setup je čas, po který musí být vstupní porty paměti stabilní před náběžnou hranou hodinového signálu a hold čas označuje časové rozmezí, po které musí vstupní porty setrvat ve stejném stavu po náběžné hraně hodinového signálu. Jeho charakterizační blokové schéma je znázorněno na obr. 4.7 a celá struktura na obr. 4.8. Vždy se charakterizuje pouze jeden signál. Hodinový signál slouží jako řízení pro CLK paměti a jeden charakterizační signál (například ME). Tyto dva signály mají v cestě vlastní nastavitelné zpoždění a modelování setup a hold času

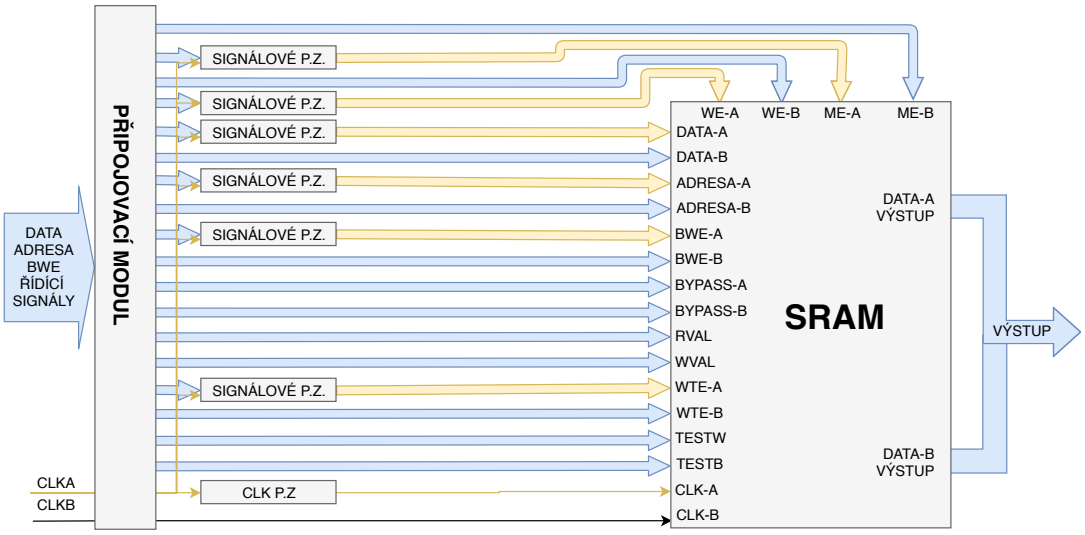
je pak záležitost rozdílu zpoždění. Správnost provedené operace se ověří z výstupu paměti na testeru. Tento postup je znázorněn na obr. 4.6.



Obr. 4.6: SHTC setup/hold



Obr. 4.7: SHTC blokové schéma pro charakterizaci setup a hold času



Obr. 4.8: SHTC-DP blokové schéma

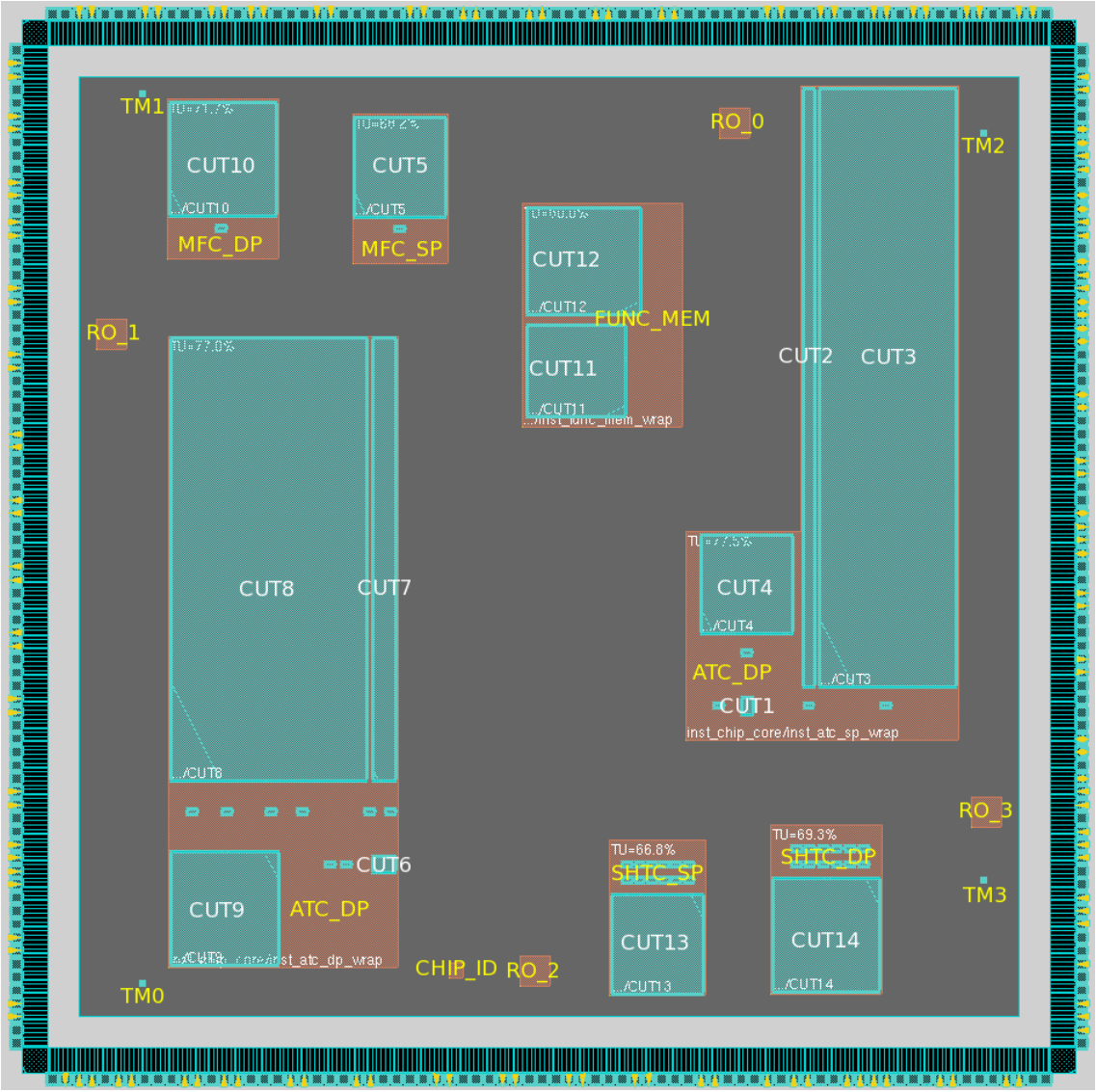
4.1.4 FUNC-MEM

Tento modul nemá žádné speciální charakterizační funkce jako předešlé moduly. Zapojení pamětí je tak podobné běžnému funkčnímu použití, ale její fyzické rozložení je otočeno o 90 stupňů. Blokové schéma modulu je znázorněno na obr. 4.9.

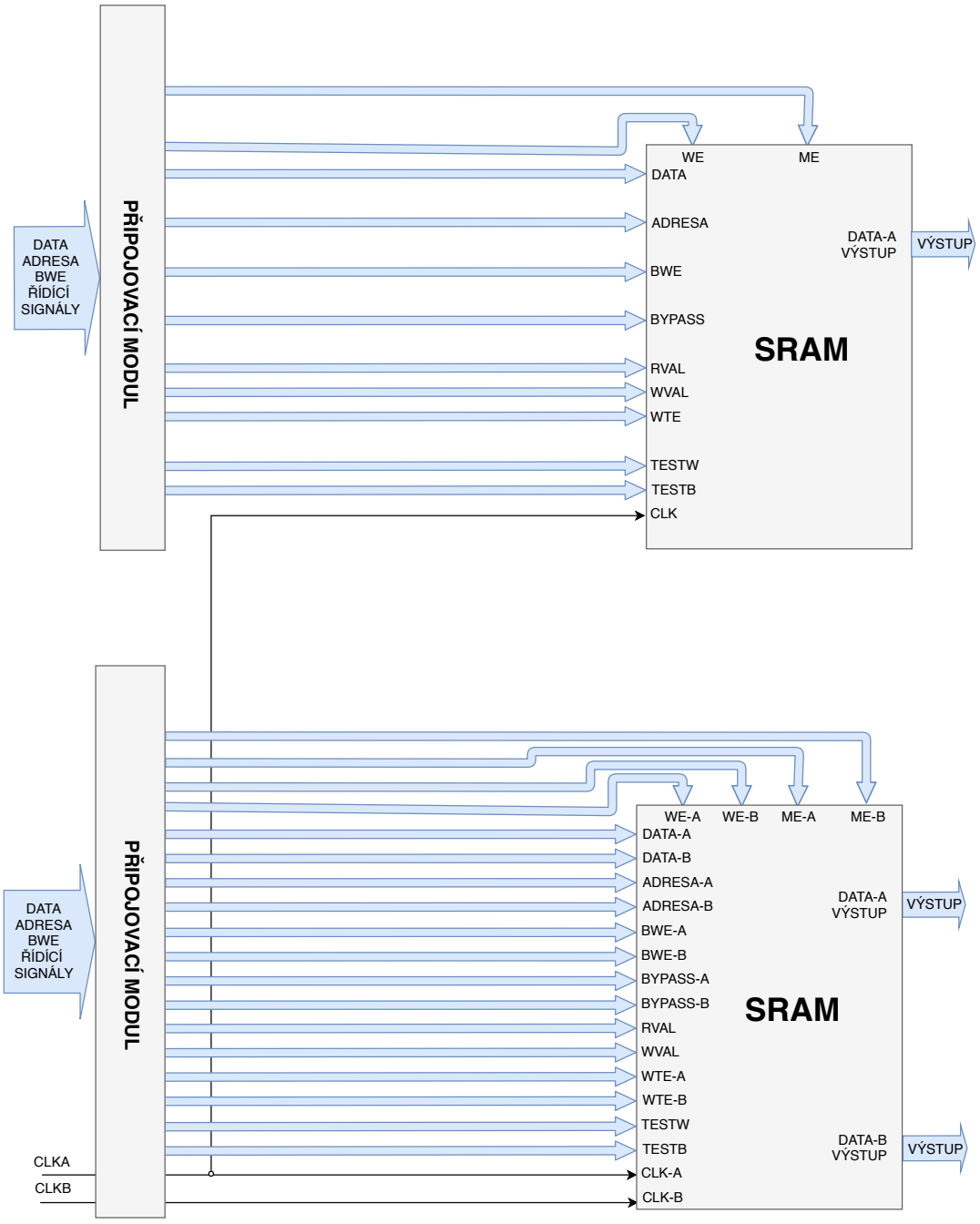
4.2 Hierarchie

Přehled testovacího čipu je znázorněn v hierarchické tabulce s počtem pamětí a jejich velikostmi, kde DP znamená dvouportová, SP jednoportová a CUT Circuit Under Test. Plán rozmístění této hierarchie na čipu je znázorněn na obr. 4.10.

- CHIP-TOP
  - ATC-SP
    - 1x CUT1 SP SRAM s 32 slovy, každé po 4 bitech, v 1 bloku
    - 1x CUT4 SP SRAM s 2560 slovy, každé po 48 bitech, v 1 bloku
    - 1x CUT2 SP SRAM s 8192 slovy, každé po 4 bitech, ve 4 blocích
    - 1x CUT3 SP SRAM s 8192 slovy, každé po 144 bitech, ve 4 blocích
  - ATC-DP
    - 1x CUT6 DP SRAM s 16 slovy, každé po 4 bitech, v 1 bloku
    - 1x CUT9 DP SRAM s 2048 slovy, každé po 36 bitech, v 1 bloku
    - 1x CUT7 DP SRAM s 4096 slovy, každé po 4 bitech, ve 4 blocích
    - 1x CUT8 DP SRAM s 4096 slovy, každé po 144 bitech, ve 4 blocích
  - MFC-SP
    - 1x CUT5 SP SRAM s 2560 slovy, každé po 48 bitech, v 1 bloku
  - MFC-DP
    - 1x CUT10 DP SRAM s 2048 slovy, každé po 36 bitech, v 1 bloku
  - SHTC-SP
    - 1x CUT13 SP SRAM s 2560 slovy, každé po 48 bitech, v 1 bloku
  - SHTC-DP
    - 1x CUT14 DP SRAM s 2048 slovy, každé po 36 bitech, v 1 bloku
  - FUNC-MEM
    - 1x CUT11 SP SRAM s 2560 slovy, každé po 48 bitech, v 1 bloku
    - 1x CUT12 DP SRAM s 2048 slovy, každé po 36 bitech, v 1 bloku
  - 4x RING-OSCILATOR (RO)
  - 4x THERMO-MONITO (TM)
  - CHIP-ID



Obr. 4.10: Rozložení na čipu



Obr. 4.9: FUNC-MEM blokové schéma

5 BIST

Built-In Self-Test (BIST) je mechanismus, který dovoluje zařízení, aby se samo otestovalo [11]. Jeho výhody spočívají ve zkrácení doby testu a zmenšení závislosti na komplexním Automatic Test Equipment (ATE) za cenu vložení testovacího zařízení do designu.

5.1 BIST pro paměti

BIST pro paměti se nazývá MBIST. Dovoluje otestovat paměť algoritmem přímo jako součást IP paměti nebo jako přidaný modul na čipu. Jeho implementace na čipu umožňuje zvýšit testovací frekvenci z důvodu kratších vzdáleností, ale tím navyšuje plochu čipu. Tuto nevýhodu lze vyvážit použitím jednoho MBISTu pro více pamětí.

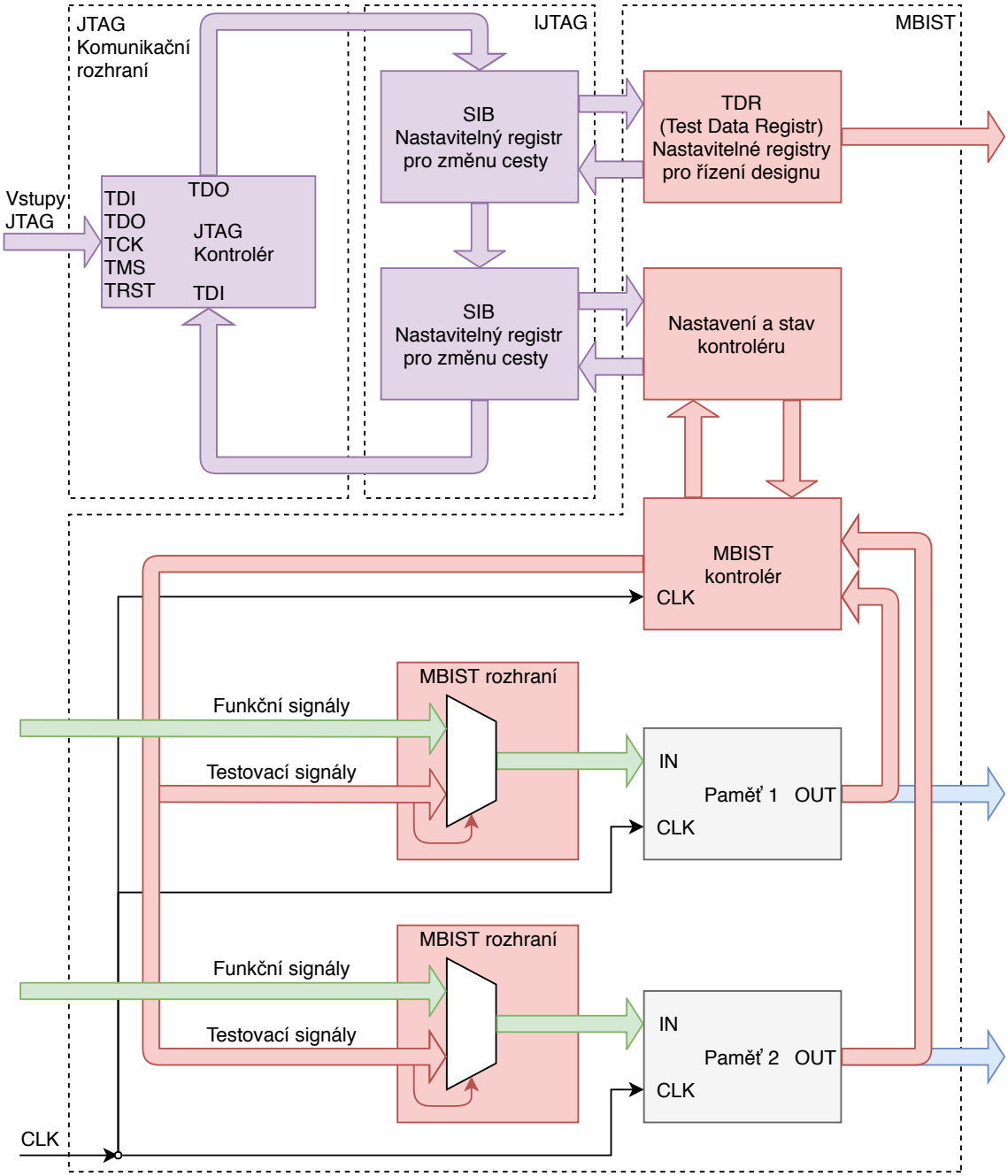
Testovací čip dovoluje komunikovat pouze s jednou pamětí současně, což limituje rychlost testování. MBIST bude vytvořen vytvořen pomocí nástroje Tessent MBIST.

5.2 Nástroj DFT pro paměti

MBIST je jeden ze softwarových nástrojů spadajících pod Tessent od Mentor Graphic. Tyto nástroje slouží pro návrh Design For Testability (DFT). Dokážou otestovat kombinační logiku, paměti a dokonce i přidat Boundary Scan napojený na JTAG. Pro účely testování pamětí nám postačí nástroj Tessent MBSIT.

5.2.1 Základní MBIST bloky

Nástroj automaticky vytváří testovací logiku, která se napojí na testované paměti v čipu, viz obr. 5.1.



Obr. 5.1: Ukázka blokově vygenerované testovací logiky MBIST a JTAG

MBIST rozhraní z obr.5.1 slouží jako přepínač mezi funkčními a testovacími signály. Tato rozhraní jsou vložena před každou testovanou paměť. Funkčním signálům toto rozhraní přidává do cesty většinou jeden multiplexor, pouze u signálů řídících paměť ještě jedno hradlo, viz obr.B.1. Mapuje se zde logický datový vstup z MBIST kontroléru na fyzický pro danou paměť a implementují se techniky na minimalizaci počtu cest mezi MBIST kontrolérem a MBIST rozhraním.

MBIST kontrolér z obr.5.1 slouží pro generaci a vyhodnocení testovacích signálů

na základě zvoleného testovacího algoritmu. Pro vyhodnocování testů se používají komparátory, které porovnávají výstupní signály z pamětí s očekávanou hodnotou z MBIST kontroléru. Každý komparátor má přiřazen registr, do kterého je zaznamenána případná chyba. Počet těchto registrů je stejný jako šířka sběrnice paměti. Za pomocí těchto registrů lze zjistit, v jaké bitové části paměti došlo k chybě. Komparátory mohou být dle nastavené umístěny v MBIST rozhraní pro minimalizaci tras nebo v MBIST kontroléru. Na konci testu se tyto registry vyčtou přes JTAG a určí, zda paměť prošla testem úspěšně. Jeden kontrolér dokáže obsloužit několik pamětí, pokud nejsou zásadně odlišné. Tělo kontroléru obsahuje instrukce algoritmu, operační set instrukcí, stavový automat, generátor adres a další konstrukce nutné k fungování MBISTu pro testování paměti.

MBIST kontrolér je řízen přes nastavitelné registry napojené na síť JTAG obsažené v modulu Nastavení a stav kontroléru z obr.5.1. Dále se zde nachází Segment Insertion Bit (SIB) registr, který po nastavení zpřístupní řetězec registrů nacházejících se v MBIST kontroléru a MBIST rozhraní.

JTAG síť je vytvářena nástrojem automaticky. Do této sítě je připojen Test Data Register (TDR) jehož vývody mění nastavení modulu tak, aby bylo možné testovat jeho paměti.

5.2.2 Práce s MBIST

Práce s MBIST může být rozdělena do dvou část:

1. Specifikace DFT hardwaru a jeho vložení do designu, viz obr.5.2
2. Vytvoření paměťových testů na základě specifikace vytvořeného DFT hardwaru

První část

Nejprve se do nástroje nahraje design, standardní buňky a paměti. Pro paměti k testování se nahrají jejich DFT šablony.

Specifikují se periody vstupních hodin modulu, jelikož při vytváření testovací logiky se nástroj pokusí z pinu paměti CLK najít cestu do těchto hodin a zjistit tak časování pro danou paměť. Pokud mu v hodinových cestách stojí překážka, (např. multiplexor) je nutné zajistiti jeho řízení vložením pomocné řídící logiky.

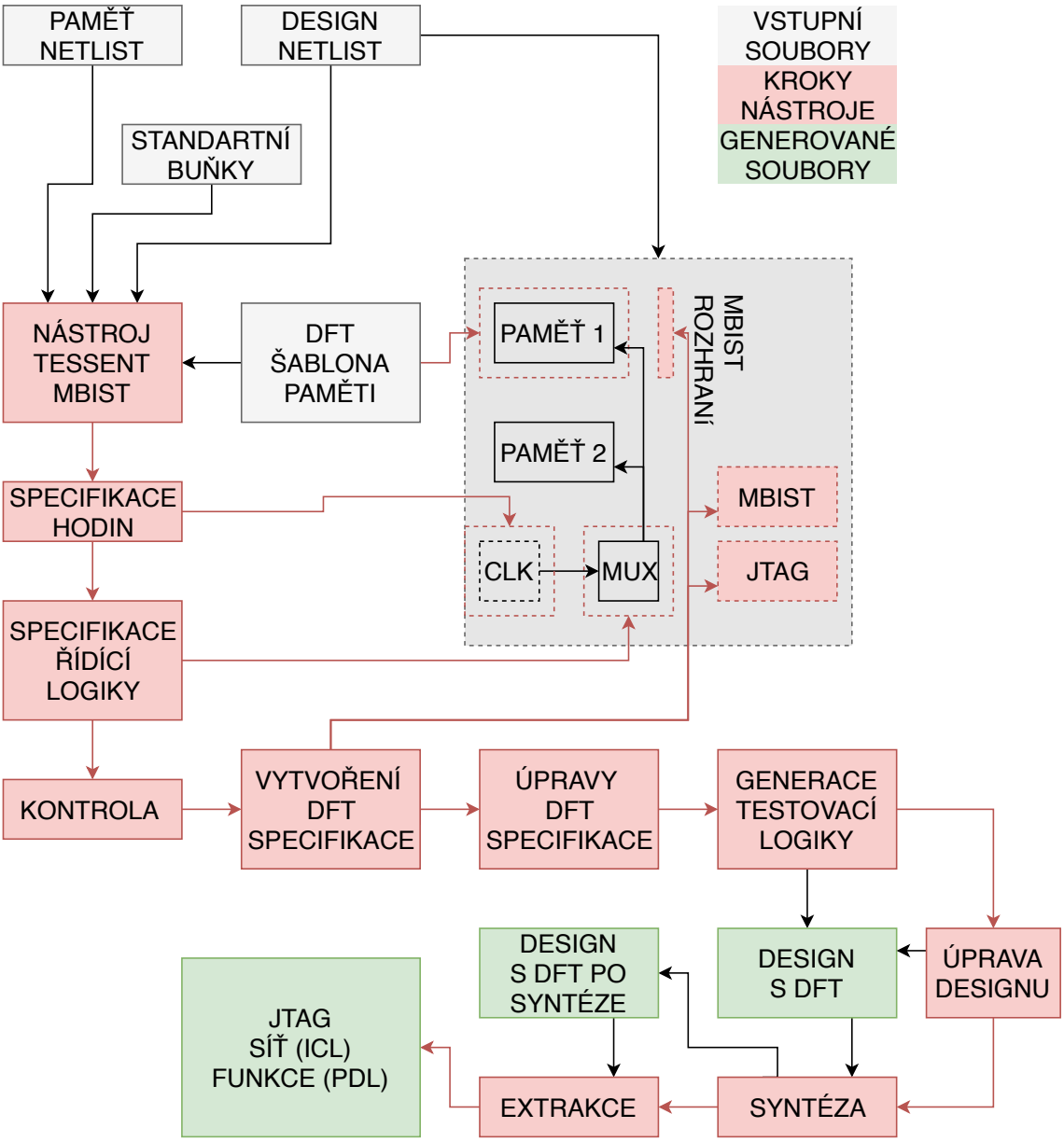
Testovatelné paměti jsou vyhledávány na základě jmen šablon pamětí a shared clusterů, které byly předem načteny. Při kontrole si nástroj ověří zda se jména a šířky portů v šabloně shodují s tím, co namapoval v designu. Tyto šablony pamětí jsou rozděleny do dvou skupin - šablony pro samostatnou paměť a šablony pro sdílenou sběrnici (shared cluster) napojenou na jednu nebo více pamětí ovládaných kontrolním portem. V šabloně pro samostatnou paměť se nachází informace o portech paměti a jejich šířce, paměťové hloubce, použitém algoritmu a fyzickém rozmístění

jednotlivých paměťových buněk, viz obr. B.2. Šablony by měla standardně dodávat firma které paměti navrhla, záleží ale na ní pro jaké softwarové nástroje jsou šablony použitelné. Více informací o těchto souborech se nachází v manuále Tessent MBIST[8]

Po kontrole je automaticky na základě nahraných souborů vytvořena DFT specifikace. V tomto kroku je možné ji zkontrolovat a upravit. Lze například definovat, jaké testovací algoritmy má MBIST obsahovat a pro programovatelný MBIST lze specifikovat maximální délku programovatelné sekvence a proveditelné operace. Je také možné nastavit i místo porovnání výstupních dat z paměti. Rozpis všech možností se nachází v návodu pro Tessent Shell[9].

Po upravení DFT specifikace ji nástroj začne generovat. Při generování testovacích modulů si program ověří, zda je v pořádku design, DFT specifikace a šablony. Nástroj automaticky vkládá testovací logiku(MBIST rozhraní) před paměti. To znamená, že pokud nástroj bude potřebovat řídit vstupní datový port modulu paměti, rozpojí jej na vstupním portu paměti a napojí výstup z vloženého multiplexoru. Je třeba dát pozor, neboť může dojít k nežádoucímu rozpojení na kritickém místě. Nástroj při generaci vytvoří RTL soubory, s vygenerovanými testovacími moduly vloženými do původního designu. Tento soubor nazve [design].vg-no-instruments. Pro jednotlivé testovací moduly vytvoří samostatné RTL soubory, které můžou být manuálně upraveny a prozkoumány. Také vznikne assembly modul, který má všechny potřebné moduly pro testovaní pamětí. Tyto moduly je možné manuálně upravit, aby vyhovovaly našim požadavkům.

Nakonec je provedena syntéza a extrakce Instrument Connectivity Language (ICL) a Procedure Description Language (PDL). Nadřazený modul tak ví, jak se připojit k podřazeným IJTAG modulům. Syntetizují se jednotlivé testovací moduly a po syntéze jsou jimi nahrazeny původní moduly, které byly v souboru [design].vg-no-instruments. Tím vznikne konečný DFT design [design].vg-full.



Obr. 5.2: Postup vložení testovací logiky pomocí Tessent MBIST

ICL popisuje propojení sítě Internal JTAG (IJTAG) a IP modulem. Standard dovoluje popsat adresovatelné registry a základní logické/mux funkce a tím dovoluje široký popis testovacího IP. Kombinace ICL popisu testovatelného IP a popisu propojení na čipu dává možnost najít každý IJTAG kontrolní/data registr, který se na čipu nachází.

PDL poskytuje informace, jak provést testy určené pro dané IP. Umístění IP nemá vliv na provedení testu. V těchto testech jsou operace zápis a čtení prováděny na registrech nebo pinech, které jsou popsány v ICL. Operace jsou časově seřazeny tak, aby vytvořily test pro IP. V těchto PDL souborech muže být celá sada procedur na otestování funkčnosti daného IP. Může též obsahovat testovací funkce s nasta-

vitelnými proměnnými. Tato kombinace dvou metod umožňuje aby navržené testy bylo možné použít na IP bez ohledu kdo a kde ho používá. Se spojením s IJTAG se také nemusí starat o napojení daného IP, jelikož se automaticky vygeneruje.

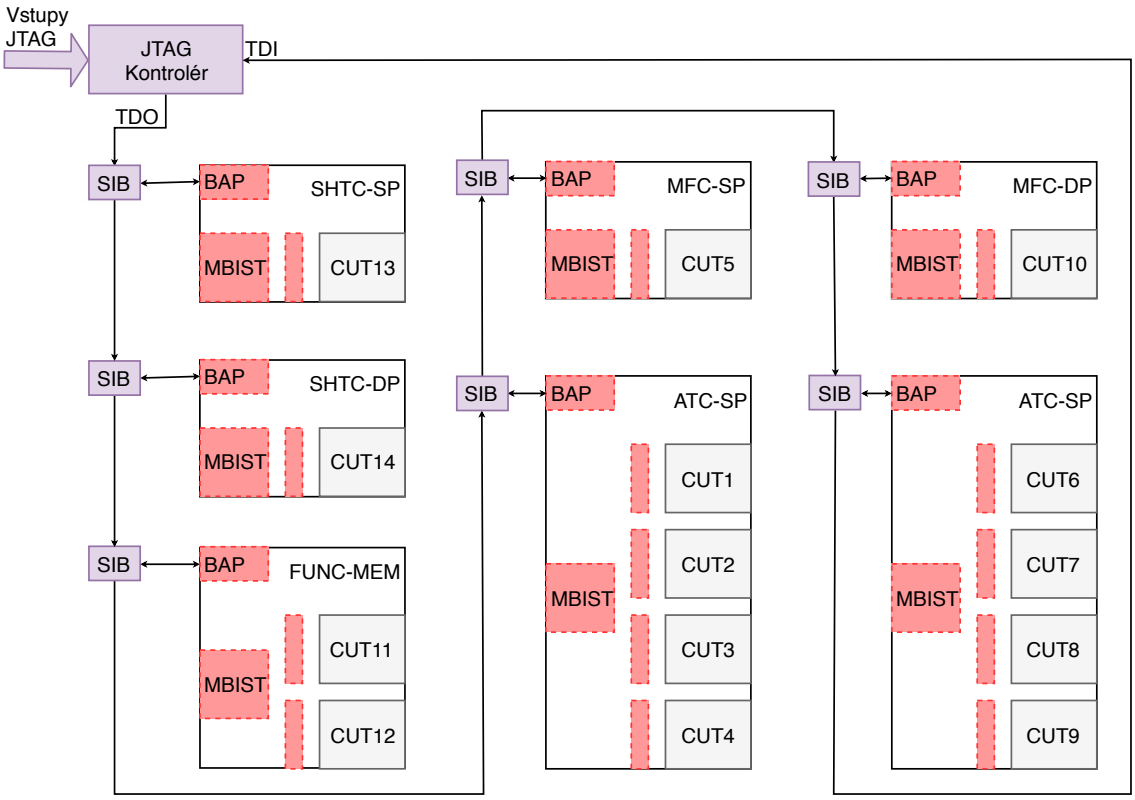
Druhá část

Pro vytváření testů je nahrán design, do kterého byla vložena testovací logika. Nástroj vytvořil složku v minulém kroku, kde jsou tyto informace uchovány. Příkazem jsou nahrány. Následně nástroj vygeneruje testovací specifikaci. Tu lze vytvořit automaticky, a to buď signoff nebo manufacture na základě informací, které získal v první části. Pokud je zvolena signoff, vytvoří se sada testů na otestování IJTAG a MBIST. Ty zkontrolují správnou činnost vložené logiky. U paměti vyzkouší pouze pár adres. Tím si ověří správnou činnost, což urychlí čas simulace. Druhý automatický test manufacture slouží pro plné testování. U těchto specifikací testů lze zvolit jejich nastavení, např. průběh diagnostiky - zda se má MBIST kontrolér zastavit při chybě či ne. Tyto úpravy lze provádět přes interaktivní GUI nebo příkazy. Pro vytvoření testů se zkontroluje, zda jsou tyto požadavky možné, a následně a následně jsou vygenerovány řídící soubory pro testery nebo simulační soubory. Pokud jsou vytvořeny simulační soubory, lze pomocí příkazu automaticky provést simulaci. Druhou možností je vytvářet testy polomanuálně. Hotový design zná všechny registry modulů, které jsou napojeny na JTAG a jsou zapsané v ICL souborech a aplikovatelné funkce na tyto moduly jsou zapsané v PDL souborech. Pokud je potřeba vytvořit test manuálně, lze využít zapisování do registrů a při tom používat funkce na automatické generovaní JTAG příkazů. Třeba pro spuštění MBISTu se zadanými parametry do funkce MBIST-RUN. Takto napsaný test pak lze převést na formát vhodný pro tester.

6 NÁVRH DFT PRO TESTOVACÍ ČIP

6.1 Požadavky

U čipu s vloženými DFT strukturami očekáváme, že MBIST provede testovací algoritmus a vyhodnotí funkčnost pamětí. Pro každý modul s pamětmi bude požadován samostatný MBIST. Pokud by se použil MBIST pouze jeden, nastaly by problémy s layoutem a napájením. Jako příklad lze uvést situaci, kdy je jeden MBIST vložen do středu čipu z obr. 4.10. Layouter by v takovém případě musel vést signály přes velké vzdálenosti ke všem pamětem. Musel by též zajistit zajistit, aby byl rozvod hodin pro všechny paměti a MBIST stejný. Dalším problémem, se kterým by bylo nutné se vypořádat, jsou různá napájení pro moduly a jejich měření spotřeby. Kvůli těmto aspektům designu je předpokládáno 7 MBISTů. Při jejich vytváření se pokusíme vyhnout změnám původní architektury designu, a to kvůli přizpůsobení pro nástroj Tessent MBIST. Dále musíme zachovat neporušené či minimálně změněné charakterizační funkce pamětí. Vložená testovací logika bude řízena přes Joint Test Action Group (JTAG), který automaticky vytvoří nástroj Tessent MBIST. MBIST kontrolér bude mít v sobě jeden algoritmus a to SMarchCHKBvcd. Algoritmus bude přejat ze šablon pro dané paměti a je podporovaným nástrojem a tudíž nemusíme implementovat vlastní. Na výstupech pamětí nechceme žádná hradla v cestě, proto pro všechny MBIST rozhraní vypneme vytváření bypass struktury okolo paměti. Tento krok nám zajistí, že na výstupu pamětí nebudou žádné multiplexory. Ostatní MBIST nastavení necháme v původním stavu. Výsledná testovací struktura by měla vypadat jak na obr. 6.1. Moduly budou připojeny přes buňky SIB, které po nastavení otevřou cestu do požadovaného modulu. To umožní rychlejší přístup do testovacích registrů z důvodu kratších řetězových cest.



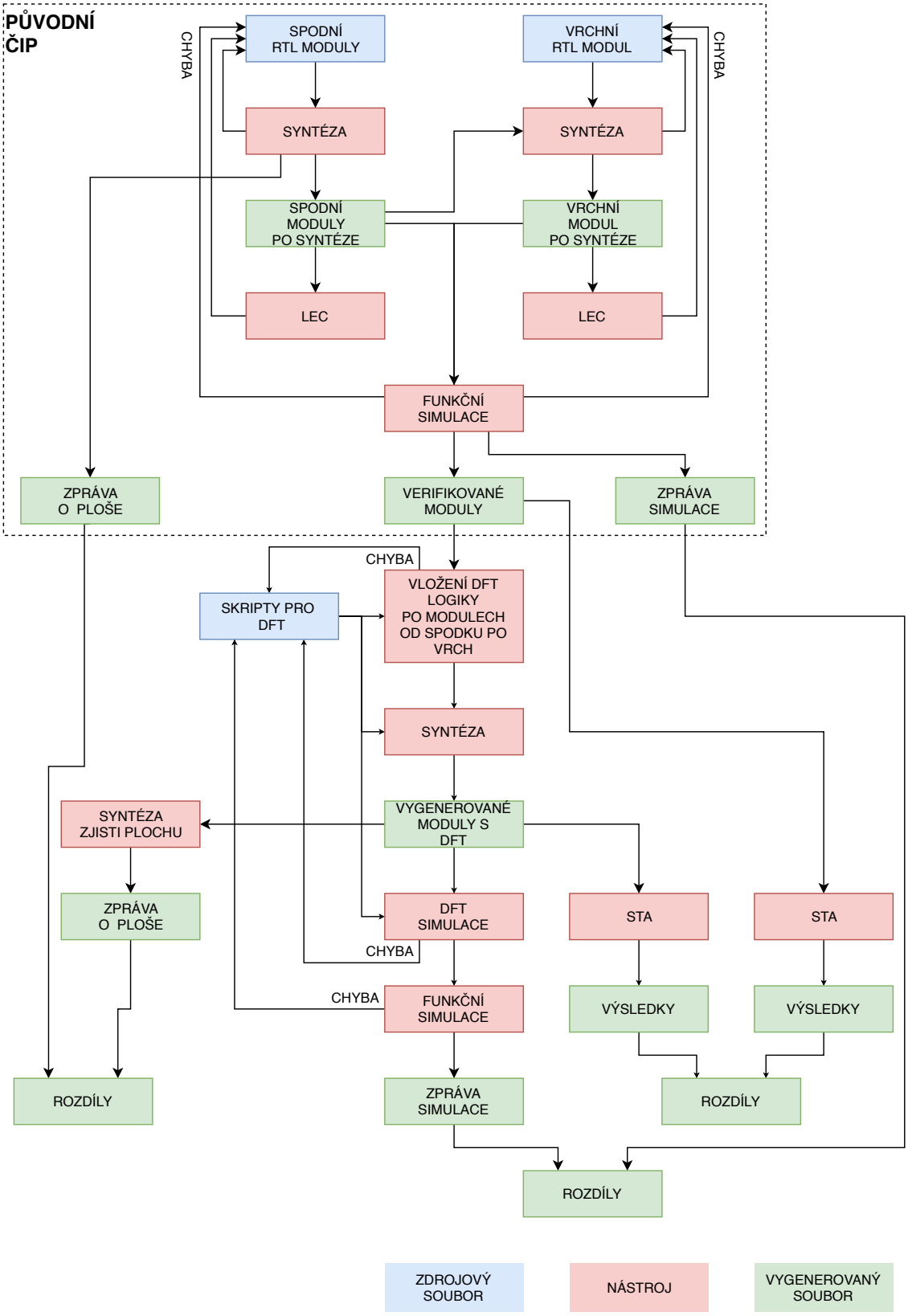
Obr. 6.1: Návrh DFT topologie

6.2 Flow

Flow popisuje, jaký je postup návrh u čipu od začátku až do dosažení požadovaného stavu. V tomto případě popisuje, jak z RTL kódu bude vytvořen funkční a verifikovaný gate level netlist a jakým způsobem je následně vkládána testovací logika. Jeho výstupem je verifikovaný gate level netlist s testovací logikou, který je připraven na layout. Toto flow je znázorněno na obr. 6.2.

6.2.1 Implementace

Implementace bude prováděna po modulech. Čip byl navržen po modulech takzvanou Bottom-Up metodou a následně po modulech syntetizován. Je tudíž vhodné použít tuto stejnou metodu pro vkládaní DFT logiky. Pokud by se tato implementace prosadila, byla by integrace DFT do takového návrhu jednoduchá. Testovací logika pamětí bude vkládána do modulů po syntéze a její verifikaci, které jsou ve formátu gate level netlist. Pokud bude vyžadována nutná změna v Hardware Description Language (HDL) kódu z důvodu přizpůsobení modulu nástroji Tessent


PŮVODNÍ ČIP
SPODNÍ RTL MODULY
VRCHNÍ RTL MODUL
CHYBA
CHYBA
SYNTÉZA
SYNTÉZA
SPODNÍ MODULY PO SYNTÉZE
VRCHNÍ MODUL PO SYNTÉZE
LEC
LEC
FUNKČNÍ SIMULACE
ZPRÁVA O PLOŠE
VERIFIKOVANÉ MODULY
ZPRÁVA SIMULACE
CHYBA
SKRIPTY PRO DFT
VLOŽENÍ DFT LOGIKY PO MODULECH OD SPODKU PO VRCH
SYNTÉZA
SYNTÉZA ZJISTI PLOCHU
VYGENEROVANÉ MODULY S DFT
ZPRÁVA O PLOŠE
DFT SIMULACE
STA
STA
CHYBA
CHYBA
FUNKČNÍ SIMULACE
VÝSLEDKY
VÝSLEDKY
ROZDÍLY
ZPRÁVA SIMULACE
ROZDÍLY
ROZDÍLY
ZDROJOVÝ SOUBOR
NÁSTROJ
VYGENEROVANÝ SOUBOR

Obr. 6.2: Flow

MBIST, bude nutné celý design znovu syntetizovat, ověřit pomocí Logic Equivalence Checking (LEC) a verifikovat funkčními testy. Každý modul bude mít svůj skript pro vkládání DFT, postupovat se bude od spodních modulů k vrchním. Flexibilita této metody spočívá v tom, že není potřeba vkládat DFT do všech modulů. Vždy stačí pouze modul topový, ke kterému můžeme přidat jakýkoliv spodní, jelikož nástroj moduly dokáže přes síť IJTAG automaticky napojovat. Jednotlivé moduly projdou několika kroky při vkládání DFT. První krokem je vložení DFT logiky do netlistu pomocí nástroje Tessent MBIST a jeho uložení. Následuje náprava nechtěných změn za pomocí příkazů v uloženém souboru. Na závěr je zavolán programu DC Shell(syntéza) z nástroje Tessent MBIST. Po dokončení syntézy budou vytvořeny automatické verifikační testovací vektory pro DFT, které budou spuštěny. Pokud verifikace neproběhne v pořádku, je nutné najít příčinu a upravit skript.

6.2.2 Verifikace

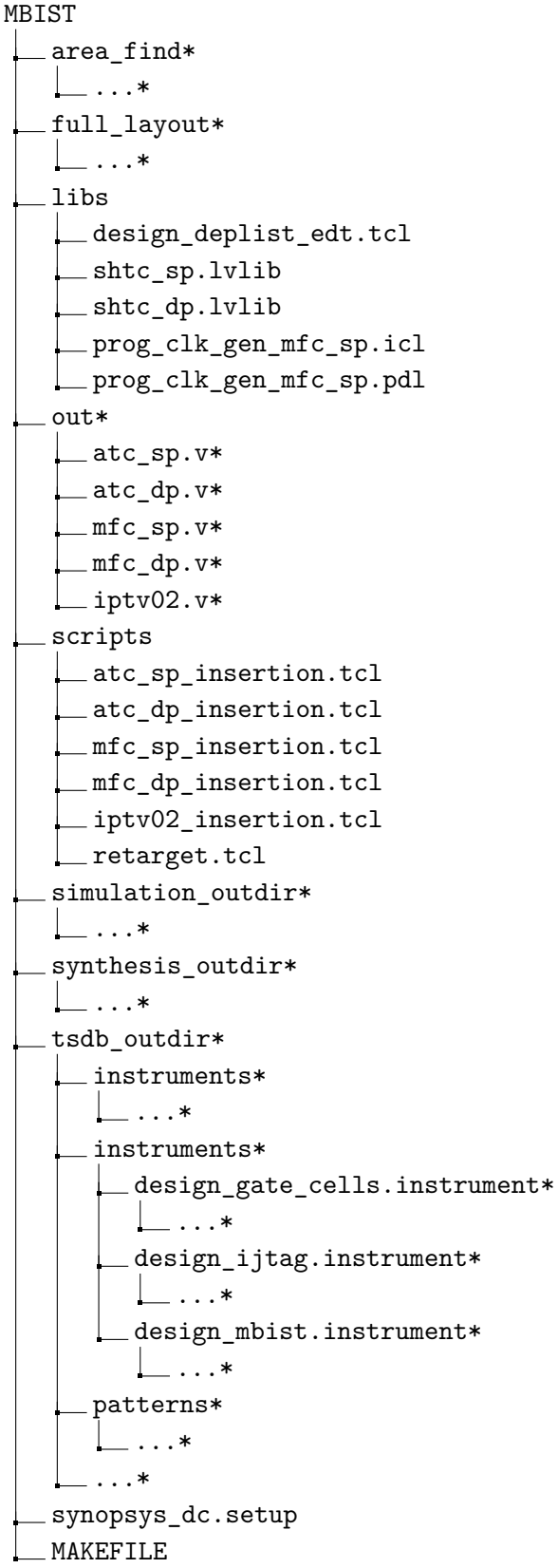
Ve flow z obr. 6.2 jsou použity 3 druhy verifikace - formální, logická simulace a statická časová analýza. Formální verifikace se provádí na původním čipu mezi RTL moduly před a po syntéze. Logická simulace se budu provádět na gate levelu netlistu pro MBIST a funkce původního návrhu. Logická simulace MBISTu je prováděná ihned po vložení testovací logiky, kdy nástroj Tessent MBIST automaticky vytvoří testy pro vloženou logiku. Následně otestuje IJTAG síť a spustí zkrácený test paměti. Tyto testy jsou aplikovány přes porty JTAGu na testovaném modulu. Logická simulace zajistí, že skript pro daný modul vytvoří funkční testovací strukturu paměti. U funkční logické simulace na čipu nejsou testy napsány pro spodní moduly, ale jen pro vrchní. Pokud je hotový jeden modul s testovací logikou a chceme ho funkčně simulovat, je nutné nahradit původní modul novým a spustit simulaci na čipu. Nastane však problém, jelikož spodní modul nemá na IJTAG pinech žádný vstup a MBSIT logika je v neznámém stavu. Kdyby se tento modul dal simulovat na úrovni modulu, daly by se tyto nezapojené piny nastavit. V prvním případě nezbývá nic jiného, než změnit netlist a manuálně napsat hodnotu, což ale není jednoduše opakovatelné. Druhá možnost spočívá v tom, že je dodatečně provedeno vložení sítě JTAG do vrchního modulu a následně provedena verifikace. V této práci je nejdříve vytvořen celý čip s DFT a poté spuštěna funkční logická simulace. Ta je řízena přes Makefile, kterému bylo nutné nachystat seznam odkazů na moduly, na nichž bude povedena. Pro zkrácení času funkčních simulací byla použita síť výpočetních strojů. Výsledky funkční simulace budou porovnány s původními výsledky za účelem zjištění, zda nedošlo ke změnám čí chybám. Pokud dojde k chybám, bude nutné je opravit, a pokud došlo ke změně, zjistit, zda tato změna něco neovlivňuje. Po logické simulaci následuje Static Timing Analysis (STA). To je řízeno přes Ma-

kefile, kterému je zadávána teplota a napětí. Skript pro nahrávání designu je nutné upravit tak, aby podporoval nový design s DFT. Dále musela být přidána omezení při analýze tak, aby JTAG piny byly v nule. To zajistí, aby MBIST logika byla ve vypnutém stavu a mohla být analyzována funkční část. Analýza je zaměřena na nové případy porušení časové specifikace, které nebyly v původním designu. Hledá se, zda nebyly vloženy hradla do kritických cest a pokud ano, je nutné najít řešení, jak je odstranit. Dále je ověřeno, zda nevznikly velké časové rozdíly v kritických cestách modulů. Kritické cesty jsou znázorněny žlutou barvou na obr. 4.3, 4.5 a 4.8.

6.3 Použití Makefilu

Makefile je strojově čitelná dokumentace, která automatizuje pracovní postup (flow) [3]. Tento design čipu používá několik speciálních makefilů pro nástroje a pak makefily, které je ovládají. Tím je dosažena automatizace flow celého čipu.

V této práci slouží Makefile jako konstruktor pro vkládání DFT do designu. Jednotlivé moduly jsou závislé na modulu skriptu určeného pro nástroj Tessent MBIST, na původním designu a na standardních buňkách. Na obr. 6.3 vidíme složku Makefile prostředí, kdy * značí soubory, které se vytváří, a které se také příkazem clean smažou. Hlavní složkou jsou scripts, kde se nacházejí skripty pro jednotlivé moduly, které využívá nástroj Tessent MBIST. Ve složce library se nachází knihovny pro popis pamětí a standardních buněk. Tyto knihovní soubory obsahují odkazy a byly napsány ručně. Pro účely této práce tato knihovna stačí, ale správně by měl Makefile tyto soubory vygenerovat na základě informací o projektu, jelikož se tyto informace mohou měnit a změny se musí automaticky propagovat přes celý design. Složku tsdb_out vytváří nástroj Tessent MBIST automaticky, pokud mu není řečeno jinak. Ukládá do ní všechny vytvořené soubory a změny v designu. Složky synthesis a simulation vytváří nástroj Tessent MBIST, a to při syntéze a simulaci. Jednotlivé syntézy či simulace jsou zaznamenány a nachystány na opakované spuštění, pokud jsou požadovány. Do složky out jsou ukládány moduly, které prošly DFT. Do složky full_layout Makefile poskládá hotový design s moduly po DFT a částí nezměněných modulů. Hotový design se následně může podrobit verifikaci a statické časové analýze. Ve složce calculated_area se nachází další Makefile, který vezme design ze složky full_layout a nahraje jej do nástroje syntézy, který vytvoří zprávu o velikosti designu.

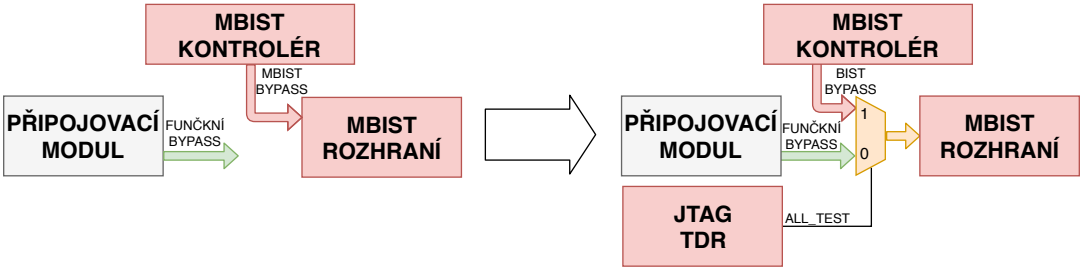


Obr. 6.3: MAKEFILE prostředí

7 IMPLEMENTACE DFT

7.1 Limitace nástroje Tessent MBIST

Při vkládání testovací struktury pro paměti pomocí nástroje MBIST byly nalezeny společné limitace, se kterými se bylo nutné vypořádat. První limitace se týká pinu BYPASS u paměti, kdy nástroj nepředpokládá, že tento pin je už používán, a automaticky ho odpojí a nahradí svým. Proto se musí modul po vložení testovací logiky upravit příkazy nebo manuálně. A to tak, aby rozpojil BYPASS pin u paměti, přidal multiplexor, napojil na něj původní a vygenerovaný BYPASS signál. Výstup multiplexoru připojil k paměti a jako řídící signál použil programovatelný registr přes JTAG, viz obr. 7.1. Ten se pak při MBISTu správně nastaví. Při resetu JTAGu se registry překlopí do takových výchozích hodnot, aby DFT logika neovlivňovala funkčnost čipu. Tato limitace musela být překonána na všech DFT modulech.



Obr. 7.1: Napojení dvou BYPASS signálů

Druhá limitace se týká řízení portů pamětí během testování, kde je zapotřebí, aby speciální testovací porty na paměti RVAL a WVAL byly nastaveny do určitých stavů. Nástroj umožňuje 1bitové porty řídit do 1 nebo 0 a při portech větších jak 1bit nastavit všechny bity do nul nebo jedniček. Pokud je ale potřeba nastavit 3bitový port do hodnoty dva, nastává problém. Na podpoře tohoto nástroje najdeme, že je to žádaná vlastnost kterou možná v budoucnu implementují. Zatím ale doporučují vytvořit obálku, která rozdělí port s širokou sběrnicí na jednotlivé porty. Doporučená metoda softwarovým výrobcem ale přináší práci na změnu šablony paměti, vytvoření nové obálky a integraci této obálky do designu, které vedou ke změnám v modulech, kde se daná paměť používá, dále v časových požadavcích pro časovou analýzu a ve verifikaci, pokud daný port využívá. Kvůli nestandardnímu designu se však nabízí jednodušší řešení.

Ve všech modulech, kam se vkládá MBIST, se nachází připojovací modul, který má za úkol připojit pouze zvolenou paměť. Pokud paměť zvolená není, jsou do paměti nastaveny výchozí hodnoty, které při MBISTu požadujeme. V připojovacím modulu, jako na obr. 7.3 od ATC, se nachází multiplexory, které mají na vstupu

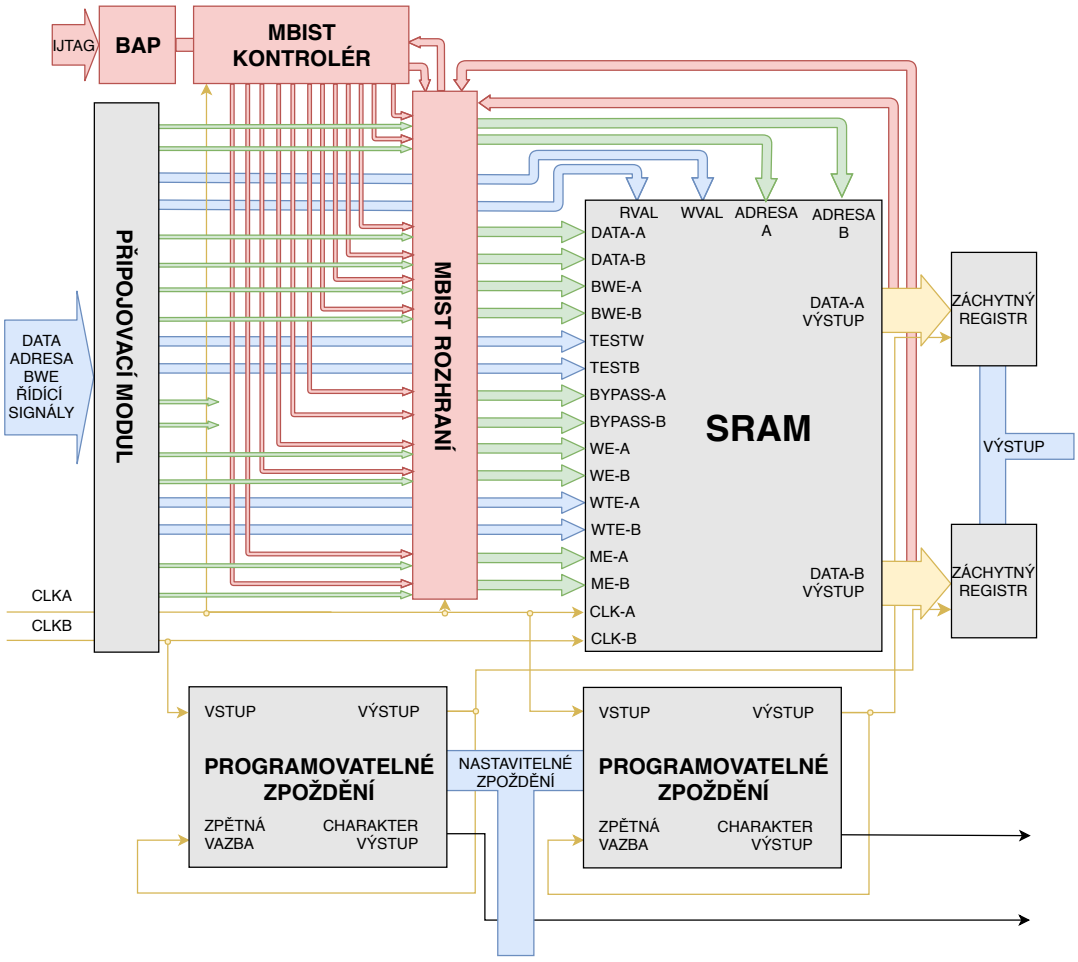
funkční signály a na druhém výchozí hodnoty. Multiplexory řídí pouze jediný signál, a pokud by tento signál šel zachytit a ovládat přes JTAG registr, snížil by se tak počet buněk, které by jinak nástroj musel vytvořit pro řízení těchto portů do výchozích hodnot. Zachycení těchto signálu ale není vždy jednoznačné. Jelikož se pracuje na souborech po syntéze, řídící signály nemusí být k nalezení, mohly být přejmenovány nebo optimalizovány. Proto se při syntéze tyto signály označí jako neoptimalizovatelné. Ale i to nemusí stačit, pokud dané signály nejsou dostupné po evaluaci nástrojem pro syntézu. V samotném verilog kódu lze používat syntaxe generate se signály sběrnice. V takovém případě nemá nástroj syntézy jasně definováno jak dané signály pojmenovat. A tak pojmenuje pouze některé a ostatním dá náhodná jména. V takovém případě pak je nutné rozepsat jednotlivé dráty v RTL, souborech aby je bylo možné označit. Naštěstí tento řídící signál řídí hradlo AND a to hradlo další AND, které je připojené na invertor napojený na port TM. Zachycením signálu TM ANDem napojeným na JTAG TDR registr lze port nastavit do 1. Tato hodnota se překlopí na 0 a propaguje se přes všechny ANDy nezávisle na hodnotách ostatních vstupních signálů a dostává se až k řídícím multiplexorům. Tím je dosaženo nastavení výchozích hodnot do pamětí. Tato metoda byla použita pro moduly ATC-SP, ATC-DP, MFC-SP, a MFC-DP.

7.2 Moduly

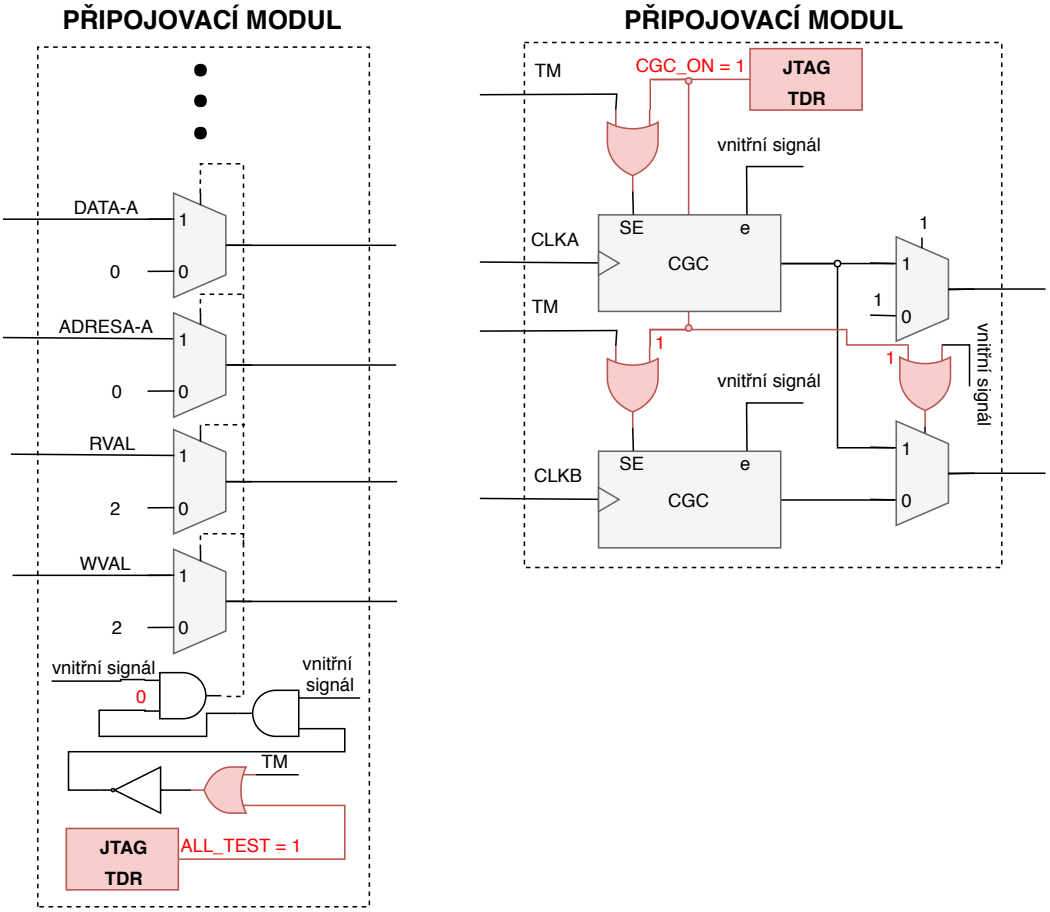
Tato část je zaměřena na změny, které nastaly při běžném postupu vkládání MBIST do těchto modulů, a problémy, které tím vznikly, a jejich řešení.

7.2.1 ATC-DP

Po vložení MBISTu do ATC-DP vzniklo blokové schéma na obr. 7.2. Z obrázku lze konstatovat že, by nemělo dojít k žádné změně charakterizačních schopností tohoto modulu, protože vložené rozhraní vkládá multiplexory do vstupních cest paměti, ale ne do výstupních. Pro kritické výstupní cesty z paměti vznikne větvení zpátky do MBIST rozhraní. To může způsobit zvětšení kapacity na výstupních vývodech a tím i zvětšení přístupové doby. Vliv bude nejspíše minimální, to ale bude ověřeno v časování. Vstupní multiplexory neovlivní parametry charakterizace protože nejsou v kritické cestě, a měření nastává, až jsou vstupní data do paměti nachystaná a validní. Do cesty hodinového signálu byla vložena odbočka pro MBIST kontrolér a MBIST rozhraní. To nevadí, protože se to následně vyřeší při vytváření rozvodu hodinových cest layouterem. Změny, které nastaly v připojovacím modulu, jsou znázorněné na obr. 7.3. V modulu se odchytily signály řídící hodinový mux a CGC a napojily na TDR.



Obr. 7.2: ATC-DP blokové schéma po vložení testovací logiky paměti



Obr. 7.3: ATC-DP připojovací modul

7.2.2 ATC-SP

Tento modul má podobnou implementaci jako ATC-DP ale zatím co modul ATC-DP, je určena pro testovaní dvouportových pamětí, ATC-DP je určen pro testování jednoportových pamětí. Jeho blokové schéma je obr. C.1. Výsledné blokové schéma po vložení testovací logiky je na obr. C.2. Jedinou zásadnější změnou bylo odebrání ovládání hodinovo multiplexoru a CGC pro druhý port. To je znázorněno na obr. C.3.

7.2.3 MFC-DP

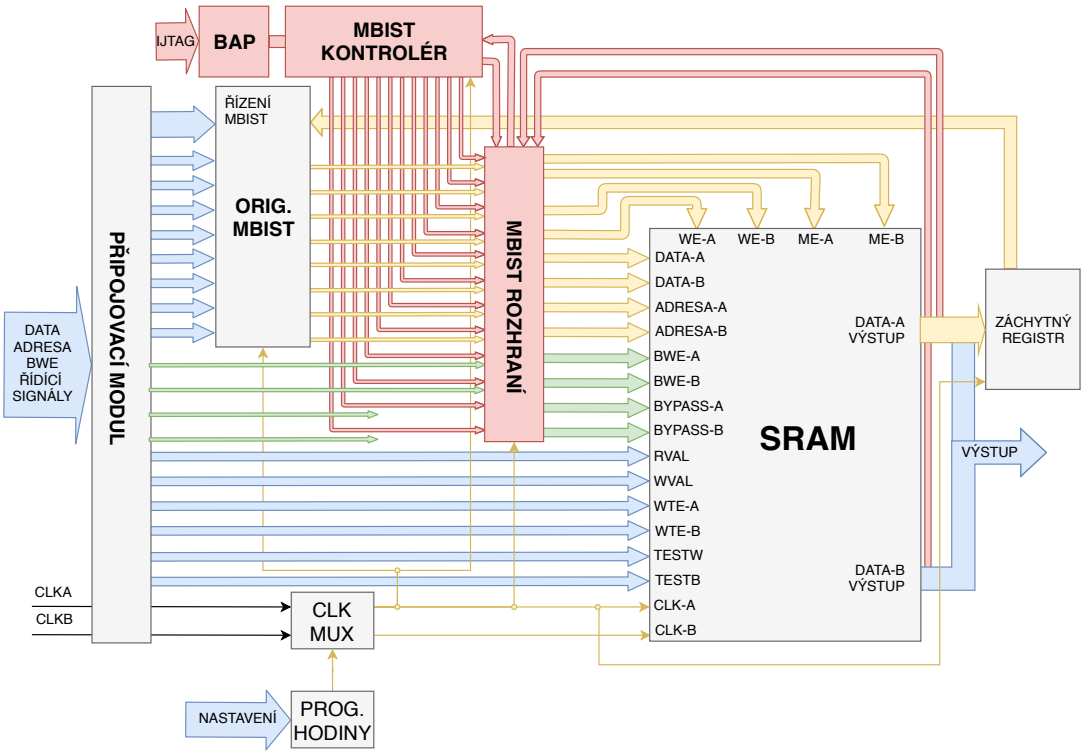
Tento modul má vlastní MBIST, který kontroluje paměť pomocí algoritmu March C. Algoritmus poskytuje dobré pokryti chyb pro účely charakterizace maximální frekvence paměti, viz tab. 3.3. Nastavení a používání tohoto interně vytvořeného MBISTu probíhá přes tester, který nahrává hodnoty do registrů, kterými je modul následně řízen. Informace o stavu testu je možné dostat na výstup přes port DOUT, který jinak slouží pro výstup z paměti. Tato metoda má výhodu v okamžitém přehledu dění v modulu.

Nastává otázka zda, je automaticky generovaný MBIST zapotřebí spolu s MBISTem původním. Generovaný MBIST by v takovém případě mohl obsahovat robustnější algoritmus pro nalezení více chyb, je ale na zvážení, zda tyto benefity převažují nad nárůstem plochy. Nejlepší možností je nahrazení původního MBISTu generovaným a zajistit, aby šel testovat na širokém spektru frekvencí. Tato možnost je výhodná, protože se nespoléhá na interně vytvořený MBIST, který se při změně paměti musí náročně předělávat, zatímco generovaný je závislý pouze na specifikaci a šabloně paměti. Budou prozkoumány oba případy.

Původní MBIST spolu s generovaným MBISTem

Při vložení MBISTu do MFC-DP na obr. 7.4 je patrné, že do kritických cest mezi pamětí a původním MBISTem bylo po vložení rozhraní přidáno zpoždění. Tím se ale zmenšila maximální frekvence na kterou paměť lze testovat. Proto je nutné dodržet několik pravidel pro zachování původních charakterizačních schopností. Zachovat minimální propagační zpoždění mezi původním MBISTem a pamětí a mezi pamětí a záchytnými registry. Minimální hodinový skew mezi původním MBISTem, pamětí a záchytnými registry.

Tato pravidla lze dodržet pokud se vygenerované MBIST rozhraní přesune před původní MBIST modul jako na obr. 7.5. Nastává problém, protože nástroj Tessent MBIST automaticky vytvoří rozhraní před pamětí jako na obr. 7.4, kterou našlo jako šablonu pro paměť. Situace se dá vyřešit několika způsoby. První možností je manuálně nebo pomocí příkazů ve skriptu přepojit signály v designu po DFT tak, aby výsledný design vypadal jako na obr. 7.5. Druhou komplikovanější možností je



Obr. 7.4: MFC-DP blokové schéma po vložení testovací logika paměti

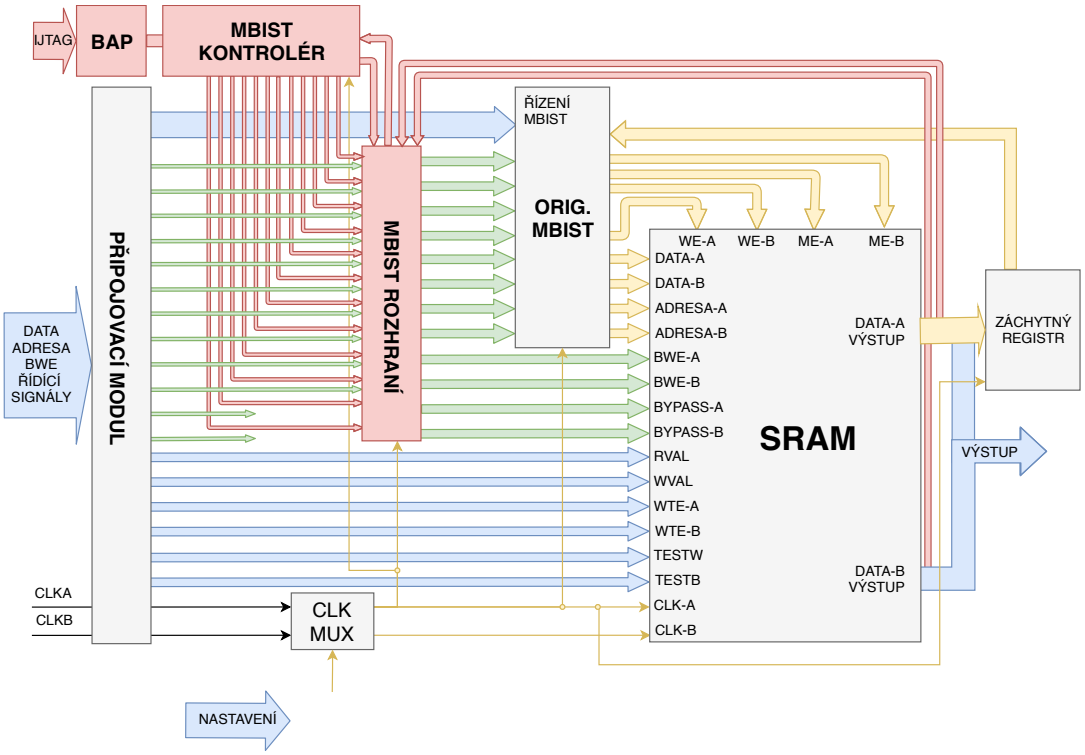
úprava designu tak, aby MBIST a paměť byla pod jedním modulem. Tento modul lze pak vydávat za paměť a tím i za šablonu. Je tak docíleno požadovaného vložení MBIST rozhraní.

Byla využita první možnost s přepojovacím skriptem a tím bylo dosaženo zapojení jako na obr. 7.5. Toto zapojení se tak vyhne vložení testovací logiky do kritických cest, jen větvení z kritického výstupu paměti do MBIST rozhraní bylo přidáno, což již bylo zmíněno v modulu ATC. Při testování budou použity externí hodiny, a proto se pro jejich nastavení přidá řídící signál napojený na TDR v připojovacím modulu z obr.7.6.

Nahrazení MBISTu

Po nahrazení originálního MBISTu by měl modul vypadat jako na obr. 7.7. Pokud bude MBIST nahrazen, je nutné zvážit několik věcí. Za prvé se musí zjistit, do jaké frekvence lze vygenerovaný MBIST použít, než začne narážet na vlastní časové limitace. Dále se musí rozhodnout, jak bude ovládán programovatelný oscilátor, zda externě nebo přes JTAG.

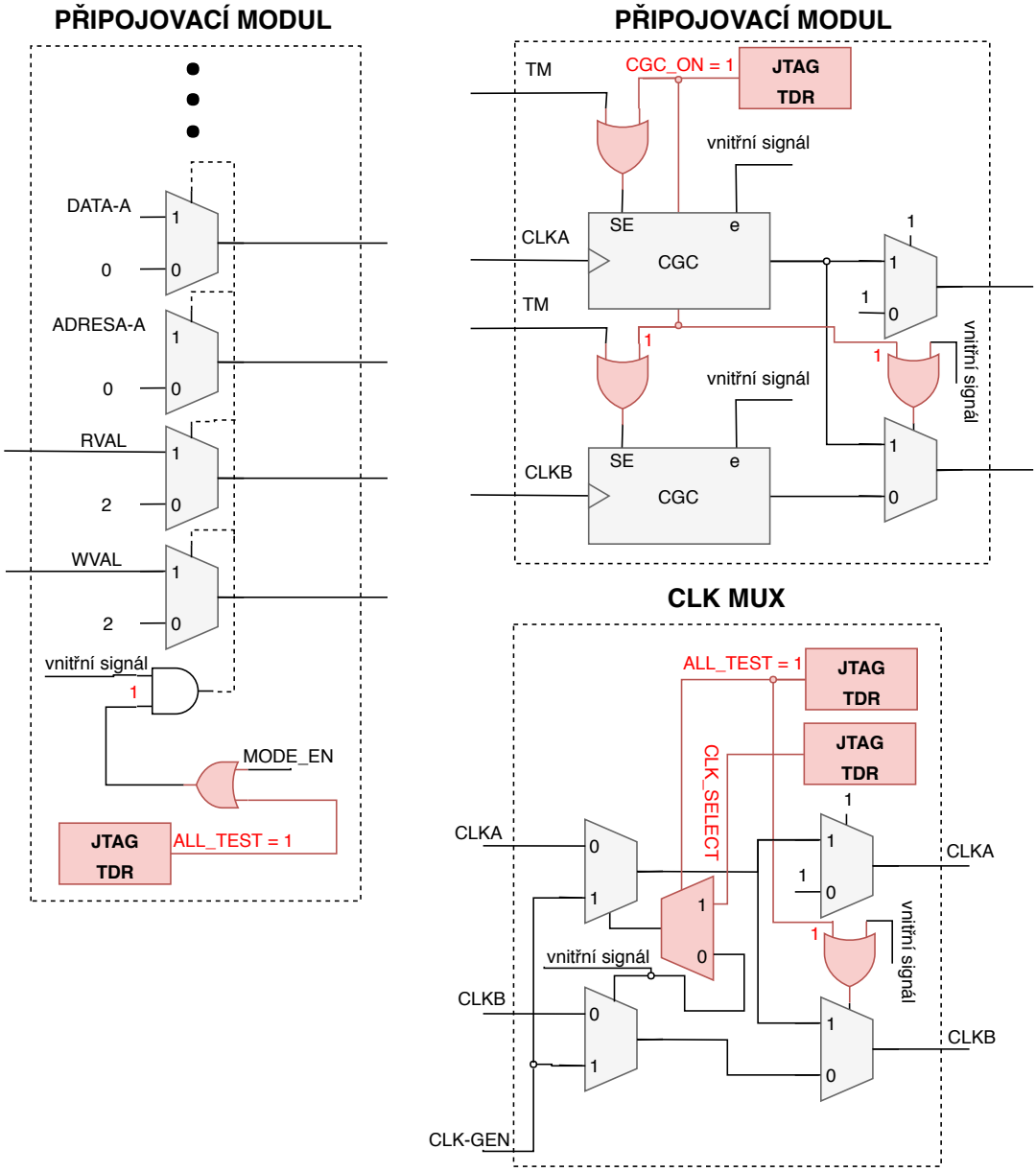
Jako první se přidá řídící logika pro multiplexor, která přepíná mezi hodinovým signálem přicházejícím externě a interním programovatelným oscilátorem, viz obr. 7.6. Na ovládání interního oscilátoru lze použít externí ATE, ale kvůli nezávislosti na ATE bylo rozhodnuto napojit oscilátor na testovací logiku řízenou přes JTAG. Proto



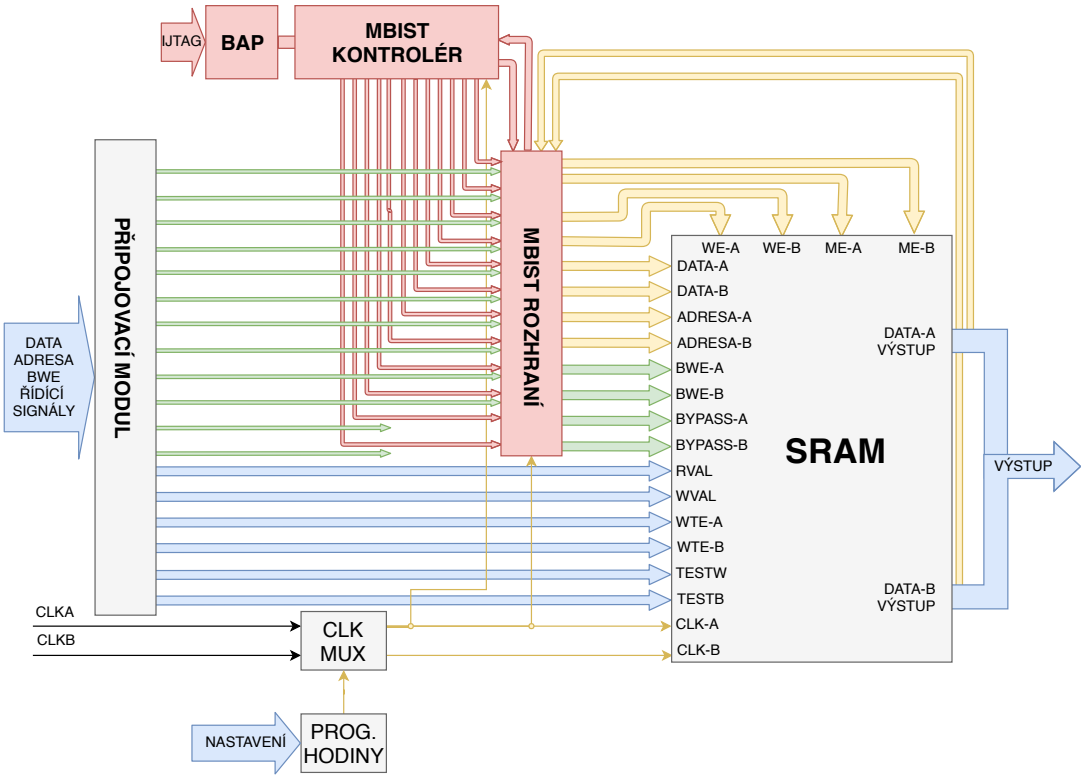
Obr. 7.5: MFC-DP požadovaná testovací logiky paměti

se vytvořil popis (ICL) tohoto programovatelného oscilátoru, který vyjadřuje, jak se má napojit na IJTAG síť. Dále se pro teto oscilátor vytvoří funkce (PDL), která po zadání hodnoty 0 až 255 nastaví příslušné registry přes JTAG do modulu oscilátoru a vypočítá předpokládanou frekvenci na jeho výstupu. Potíž ale nastává při testování v různých teplotách. Frekvence interního oscilátoru je velice teplotně závislá a funkce pro vypočítání frekvence teplotu nebere v potaz. Vypočítaná frekvence se tak liší od té reálné. Na základě vypočítané frekvence se ale vypočítává délka testu. Pokud je reálná frekvence menší než vypočítaná, tester bude chtít vyhodnotit test dřív, než skončil a nahlásí chybu. To jde vyřešit několika způsoby. Vytvořit funkci, která je teplotně závislá a vytvořit tak pro tyto teploty speciální statické testovací vektory. Nebo funkce na počítání navrhnout tak, aby za každých teplotních podmínek dávala frekvenci nižší než reálná. To má ale za následek delší průběh testů.

Další problém je jak vytvořit test pro nalezení maximální frekvence. Prvním řešením bylo vytvořit velký test, kdy se frekvence postupně zvětšovala od nejmenší po největší. To je ale časově náročná testovací operace a bylo požadováno, aby se použila původní metoda binárního vyhledávání tedy aby testy byly dynamické neboli podporovaly skoky. Jenže nástroj Tessent MBIST vytváří testy PDL úrovně 0, které jsou statické ale ne PDL úrovně 1, které jsou dynamické[6]. Testy je možné jdou konvertovat do několika formátu(WGL, SVF). Jedním řešením, jak ze statických



Obr. 7.6: MFC-DP připojovací modul



Obr. 7.7: MFC-DP nahrazený MBIST

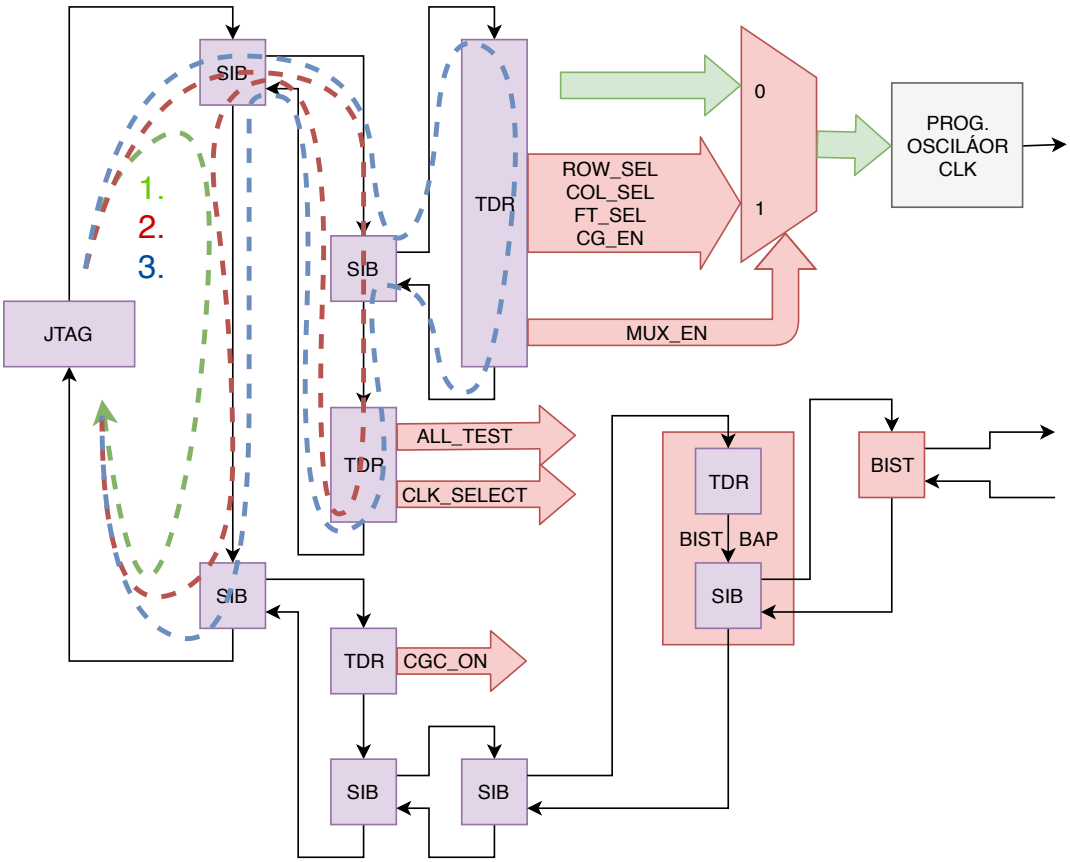
testů udělat dynamické je použití PDL interpretoru úrovně 1 a do původního PDL skoky doplnit a následně vygenerovat řídící soubor pro ATE. V tuto dobu PDL interpretor není k dispozici a musí se tak najít jiné řešení.

Bylo tedy zváženo několik přístupů. Použít zmíněný dlouhý test, kdy se v testovacím programu rozdělí test na jednotlivé části pomocí skoků a napíše se vyhledávací test. Nebo se vytvoří test na každou frekvenci a ATE software bude nahrávat test závislý na předešlém výsledku. Další možností je vygenerovat test se třemi testovacími vektory pro tři frekvence a ten následně popsat. Tři testovací vektory proto aby bylo znázorněno, z jakých částí lze skákat v testeru, jelikož IJTAG je síť konfigurovatelná a první řetězce slouží jako konfigurace sítě, viz obr. 7.8. První a druhý vektor nastavuje cestu a třetí vektor nastavuje programovatelný oscilátor. Nejde tedy skákat z vektoru jedna přímo do třetího. Z popisu vektorů tester napíše vyhledávací test.

Dále bylo třeba zjistit, jak má tester dlouho čekat do konce cestu. To lze vyřešit periodickým čtením registrů, které signalizují běh MBISTu anebo se vypočítá dostatečná čekací doba na základě počtu hodinkových cyklů algoritmu a frekvence interního oscilátoru.

Rozhodnutí

Integrace původního MBISTu spolu s generovaným MBISTem by se s velkou pravděpodobností nepoužila kvůli malým výhodám, které dva MBISTy přináší nad



Obr. 7.8: MFC-DP postup nastavení oscilátoru přes IJTAG síť

nárůstem plochy čipu. Byla tedy zvolena možnost náhrady původního MBISTu a jeho ověření pro MFC charakterizaci. S tím, že se vyzkouší s nastavitelným oscilátorem přes JTAG.

7.2.4 MFC-SP

Tento modul má stejnou funkci jako MFC-DP, ale zatímco předešlý modul byl určen pro testovaní dvouportových pamětí, MFC-SP je určen pro testování těch jednoportových. Jeho blokové testovací schéma je na obr. C.4. Při vkládání testovací struktury se potýkáme se stejnými problémy jako v modulu MFC-SP. Výsledné blokové schéma po vložení je na obr. C.5 a požadované blokové schéma na obr.C.6. Jediné dodatečné změny se týkaly odebrání řízení CGC a odebrání řízení jednoho hodinového multiplexoru, viz obr.C.3.

7.2.5 SHTC-DP

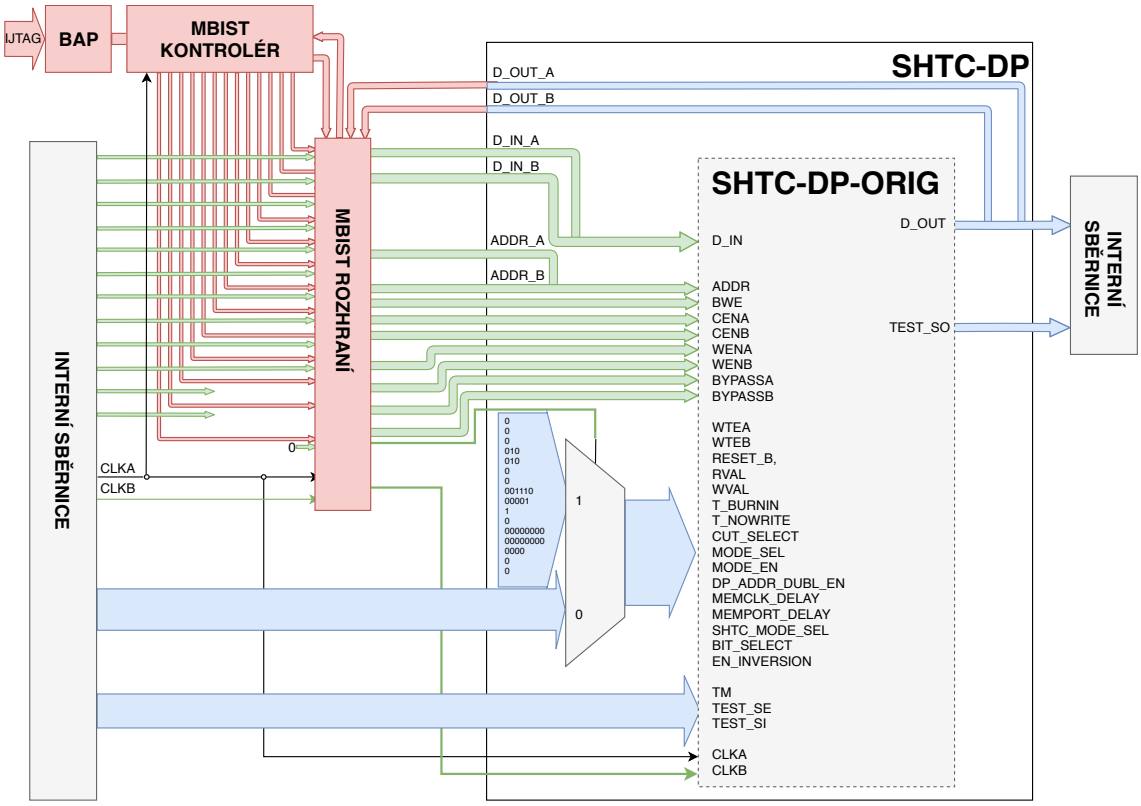
Pří normálním vložení MBISTu do tohoto modulu bylo jasné, že MBIST vloží rozhraní před paměť přímo do kritických cest pro charakterizaci. Tato změna by charakterizaci zkreslila velice. A přepojení signálu kvůli nápravě pomocí příkazů by bylo nepřehledné. Nemluvě o těžším nastavení modelu do stavu vhodného pro MBIST.

Bylo proto rozhodnuto vytvořit šablonu paměti, kdy je celý charakterizační modul pokládán za paměť. Tím se zabrání jakékoliv změně jeho charakterizačních schopností. Šablona je založena na původní paměti a přizpůsobena pro tento model. Tato možnost je realizovatelná, protože se v modulu nachází pouze jedna paměť a nástroj tak dokáže správně vytvořit jedno MBIST rozhraní.

U tohoto modulu nastal ale problém, neboť jedna vstupní a výstupní sběrnice v sobě obsahuje dva porty. Takový zápis nástroj Tessent MBIST nepodporuje a potřebujeme je rozdělit. Musela se vytvořit obálka, která má dva vstupní porty a uvnitř je napojena do původního. Dále musí mít tří výstupní porty - port a, port b a původní výstupní sběrnici. Jelikož výstupní sběrnice z modulu je větší než dva porty spojené dohromady, nejdou rozdělit jako vstupní.

Další aspekt který se musel vyřešit, bylo nastavení modulu tak, aby fungoval jako v DMA módu. Jelikož se vytváří obálka, lze ji použít a vložit do ní multiplexory s původními signály a požadovanými hodnotami pro nastavení DMA módu. Tyto multiplexory budou řízeny testovacím portem napojeným na vygenerovaný MBIST.

Při syntéze se muselo vyřešit několik problémů. Nový obal měl být poprvé novým modulem, aby se nemusel měnit původní modul, ale to bylo náročnější než se čekalo. Jelikož je design hotový, bylo pro přidání modulu nutné udělat spoustu změn v celém designu a po chvíli zkoušení se od plánu odstoupilo. Proto se v původní složce



Obr. 7.9: SHTC-DP blokové schéma po vložení testovací logiky paměti

vytvořil obal se stejným jménem shtc-sp. Původní modul se přejmenoval na shtc-sp-orig, který se se pak použil v nové obálce, aby se nemuseli měnit jména v časových požadavcích z CUT14 na shtc-sp-orig/CUT14, se modul shtc-sp-orig při syntéze rozpustil a časové požadavky se tak nemusely měnit. Tímto postupem proběhla syntéza v pořádku. Další změna byla pro nástroj LEC, kdy bylo přidáno nastavení testovacího portu do 0, tak jak při běžné funkci modulu.

Poslední změnou bylo upravení nadřazeného modulu pro správné připojení nového modulu. Byla napojena sběrnice na dva porty, přidány dva výstupní porty a do vstupního testovací portu zasazena 0 pro setrvání ve funkčním módu, dokud nedojde ke vsunutí testovací logiky. Výsledná implementace je na obr. 7.9.

7.2.6 SHTC-SP

Tento modul má stejnou funkci jak SHTC-DP, ale zatímco předešlý modul byl určený pro testování dvouportových pamětí, SHTC-SP je určen pro testování jednoportových pamětí. Byla zvolena stejná metodika jak v SHTC-SP. Jediný aspekt který se změnil je počet portů v paměti na jeden. To znamená, že v obálce se nemusela rozdělovat vstupní datovou sběrnice na dvě a stačí původní. Výstupní datová sběrnice se ale stále musí rozdělit. Jelikož sběrnice je větší než sběrnice paměti. Proto je na

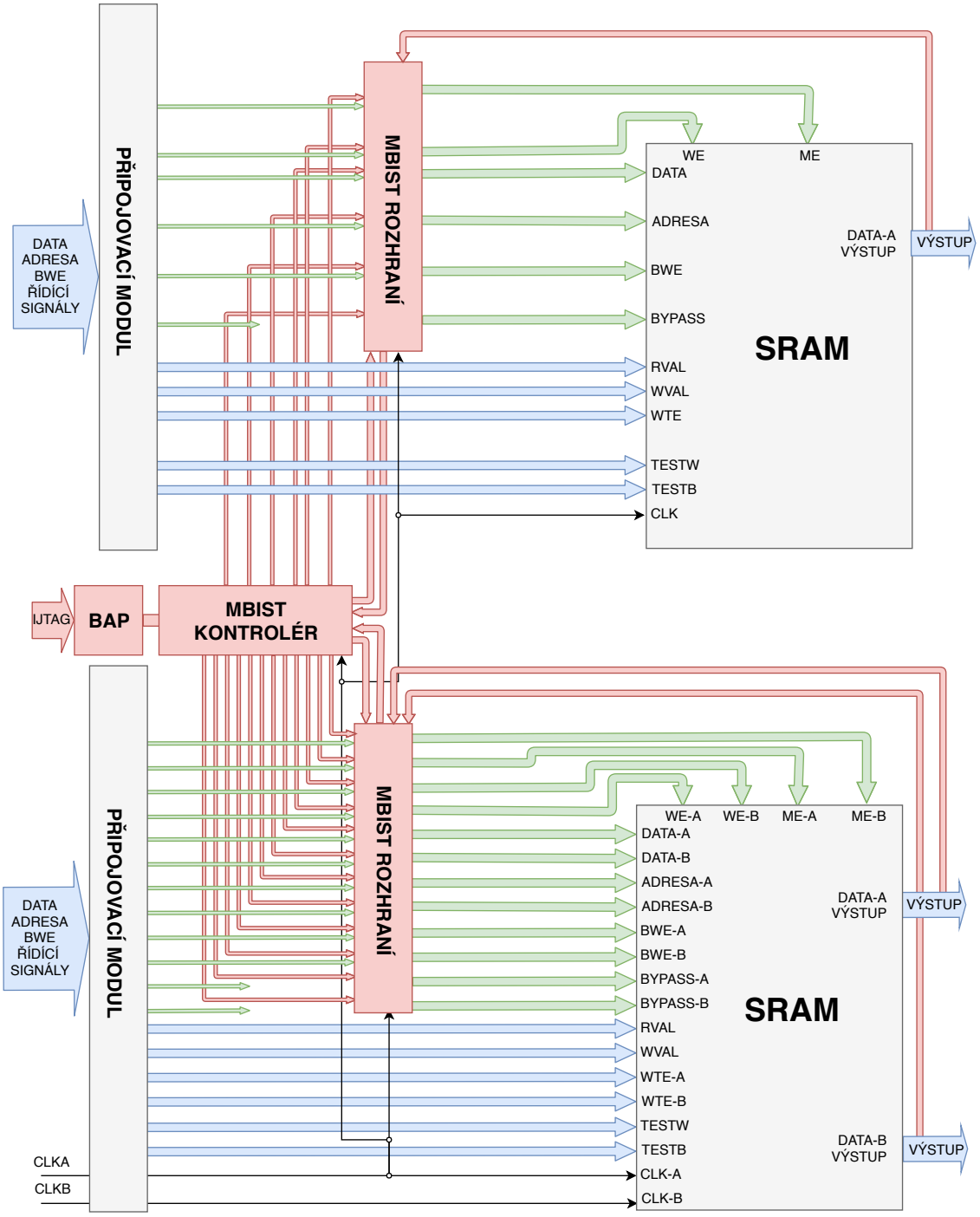
výstupu původní sběrnice a sběrnice pro MBIST. Výsledná implementace je na obr. C.8.

7.2.7 FUNC-MEMORY

Při vložení MBISTu do FUNC-MEM vzniklo blokové schéma na obr. 7.10. Tento modul nemá žádné speciální charakterizační funkce a nejsou tu žádné zásadní požadavky. Pro tyto typy pamětí je nástroj Tessent MBIST ideální. Při vytváření se bylo nutné vypořádat jen s 1. a 2. limitací nástroje.

7.2.8 PADRING

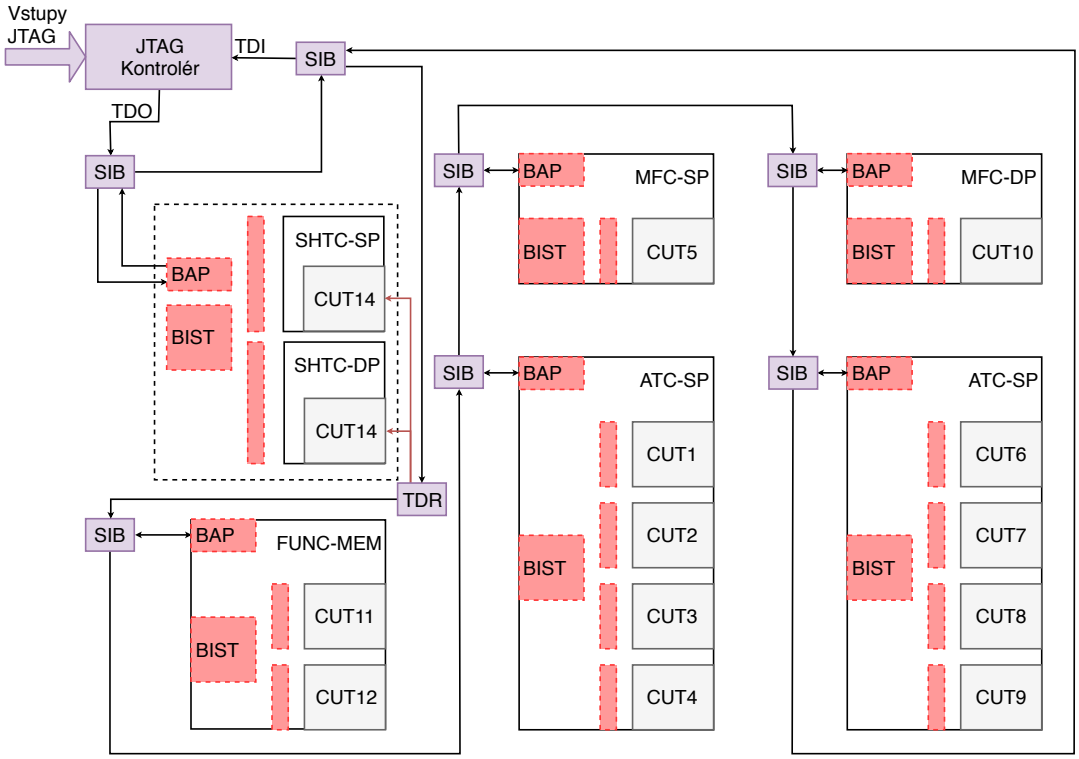
Padring obsahuje vstupní a výstupní buňky, které slouží jako brána čipu dovnitř a ven. V tomto modulu byl změněn výstupní PAD TDO z výstupní buňky, která je vždy průchozí, na nastavitelnou. Tuto nastavitelnost pinu používá JTAG.



Obr. 7.10: FUNC-MEM blokové schéma po vložení testovací logiky paměti

8 VÝSLEDKY

Od původního DFT návrhu z obr. 6.1 se provedla jedna změna. Místo 2 MBISTů vznikl jeden společný pro moduly SHTC-SP a SHTC-DP z důvodů, které jsou popsány v implementaci. A vznikl DFT design na obr. 8.1.



Obr. 8.1: Výsledná DFT

8.1 Velikost před a po

Provedli jsme porovnání velikosti čipu před a po, které jsou znázorněné v tab. 8.1. U modulů s většími paměťmi je procento nárůstů plochy menší z důvodu většího poměru MBIST/paměť. Kdybychom chtěli redukovat plochu, snažili bychom se mít pouze jeden MBIST, to však v našem případě nebylo možné. Je také nutné podotknout, že všechny vygenerované MBIST kontroléry zabírají přibližně stejnou plochu. U modulů MFC je pozorován nárůst plochy i přesto, že originální MBIST byl nahrazen, je tedy zřejmé, že generovaný MBIST je větší než originální.

Tab. 8.1: Nárůst plochy designu

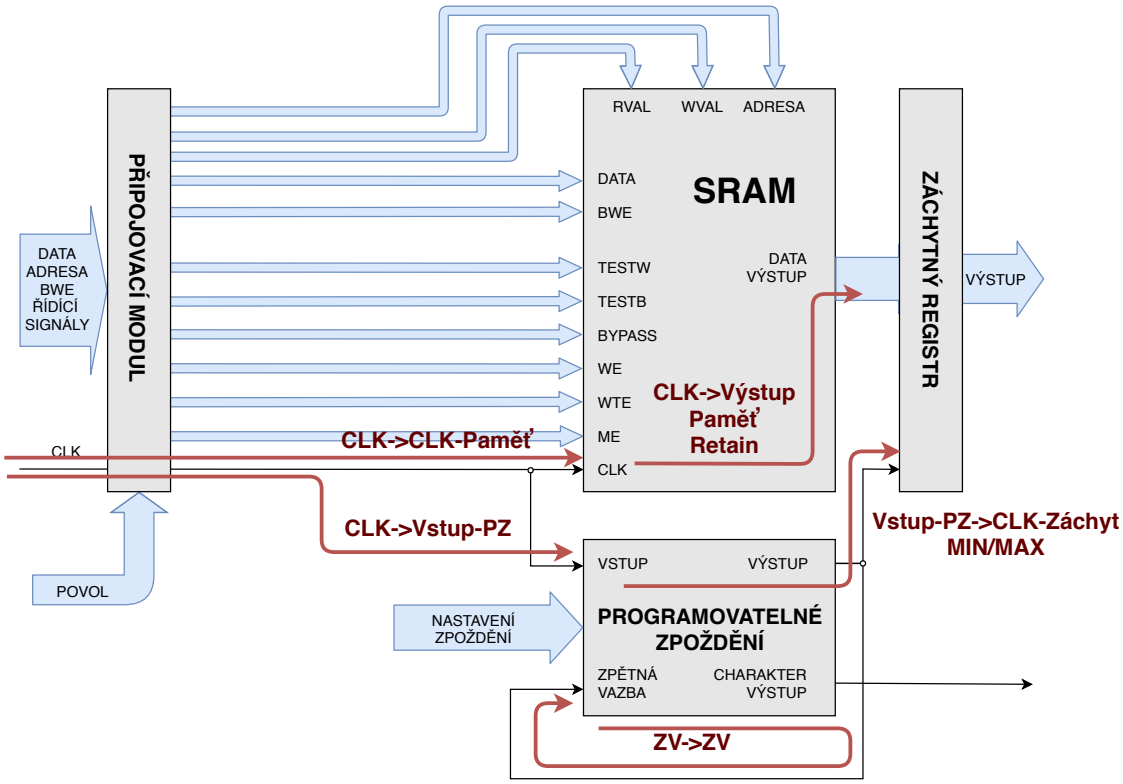
Modul	**Změna v %**
ATC-SP	1.8
ATC-DP	2.1
MFC-SP	2.07
MFC-DP	3.07
SHTC-SP + SHTC-DP	4.7
FUNC-MEM	5.7
DESIGN	2.9

8.2 Logická simulace

Funkční a DFT logické simulační verifikace proběhly v pořádku. Ve funkční simulaci nebyly nalezeny nové chyby, které by se již nevyskytovaly v té původní. V průběhu práce však tato verifikace odhalila chyby s piny bypass a také prohození portů ve vytvořené obálce pro moduly shtc-sp/dp. Změnila se však její varování. Jedná se o informativní varování, které byly lehce pozměněna od těch původních a nemají na funkci čipu vliv.

8.3 STA

V STA pro moduly ATC nebyly nalezeny chyby. Byly však nalezeny drobné změny v jeho kritických cestách znázorněny na obr. 8.2. Jejich změny jsou zaznamenány v tab. 8.2 a 8.3, kdy kladná procenta v tabulce značí narůst časové cesty a záporné zmenšení časové cesty. V tomto modulu má největší podíl na změny nárůst kapacity na výstupních pinech paměti, a to o 80%. To je způsobeno paralelním napojením na MBIST. Kritická cesta CLK z čipu do modulu není měřena, protože v teto fázi časování rozvodu hodinového signálu, tzv. clock tree, neexistuje. A jelikož práce řeší pouze front-end, není podpora pro vytvoření layoutu dostupná. Proto se cestou není nutné zabývat.



Obr. 8.2: Kritické časové cesty pro modul ATC

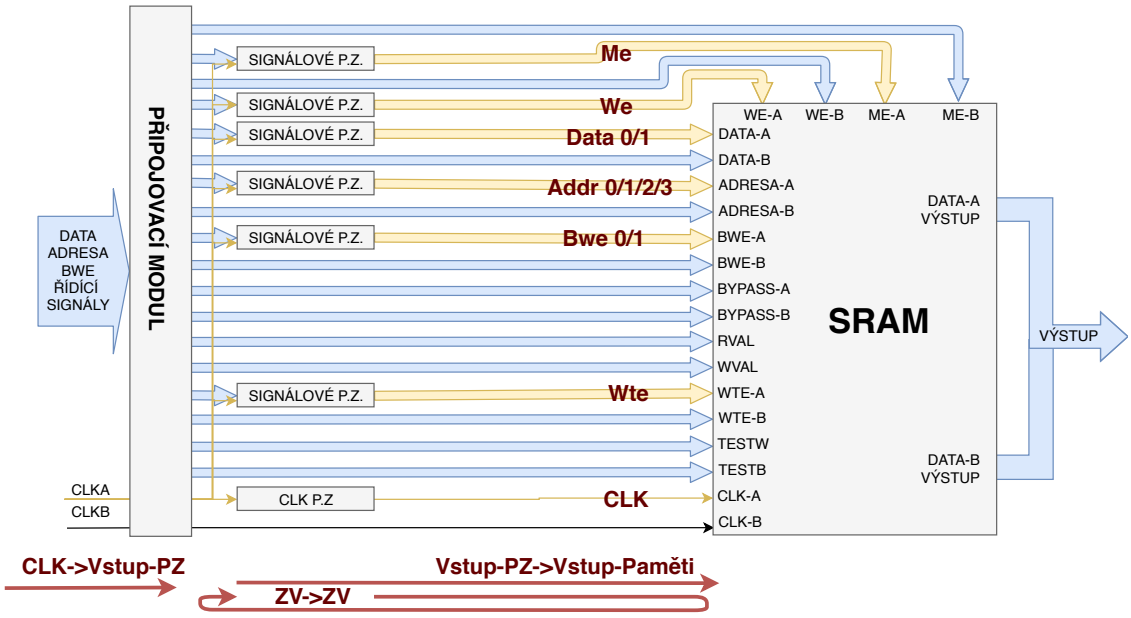
Tab. 8.2: Změny v kritických cestách pro paměti modulu ATC-SP

Cesta	**Změna v %**			
-	CUT1	CUT2	CUT3	CUT4
CLK-Paměť->Výstup-Paměť	0.37	0.23	0.23	0.26
RETAIN	0.44	0.35	0.34	0.32
CLK->CLK-Paměť	-	-	-	-
CLK->Vsup-PZ	-	-	-	-
ZV->ZV	0	0	0	0
Vstup-PZ->CLK-Záchyt-MIN	0	0.25	0	0
Vstup-PZ->CLK-Záchyt-MAX	0.22	0.23	0.20	0.20
CLK-SKEW	-	-	-	-

Tab. 8.3: Změny v kritických cestách pro paměti modulu ATC-DP

Cesta	**Změna v %**			
-	CUT6	CUT7	CUT8	CUT9
CLK-PAMET->Q-Paměť	0.35	0.20	0.19	0.31
RETAIN	0.41	0.37	0.36	0.30
CLK->CLK-Paměť	-	-	-	-
CLK->Vsup-PZ	-	-	-	-
ZV->ZV	0	0	0	0
Vstup-PZ->CLK-Záchyt-MIN	0	0	0	0
Vstup-PZ->CLK-Záchyt-MAX	0	0	0	0
CLK-SKEW	-	-	-	-

V STA pro moduly SHTC nebyly nalezeny chyby, jelikož se implementace vyhnula vložení testovací logiky přímo do modulů SHTC-SP a SHTC-DP. Kritické časové cesty modulu jsou na obr.8.3 a jejich změny v tab.8.4.



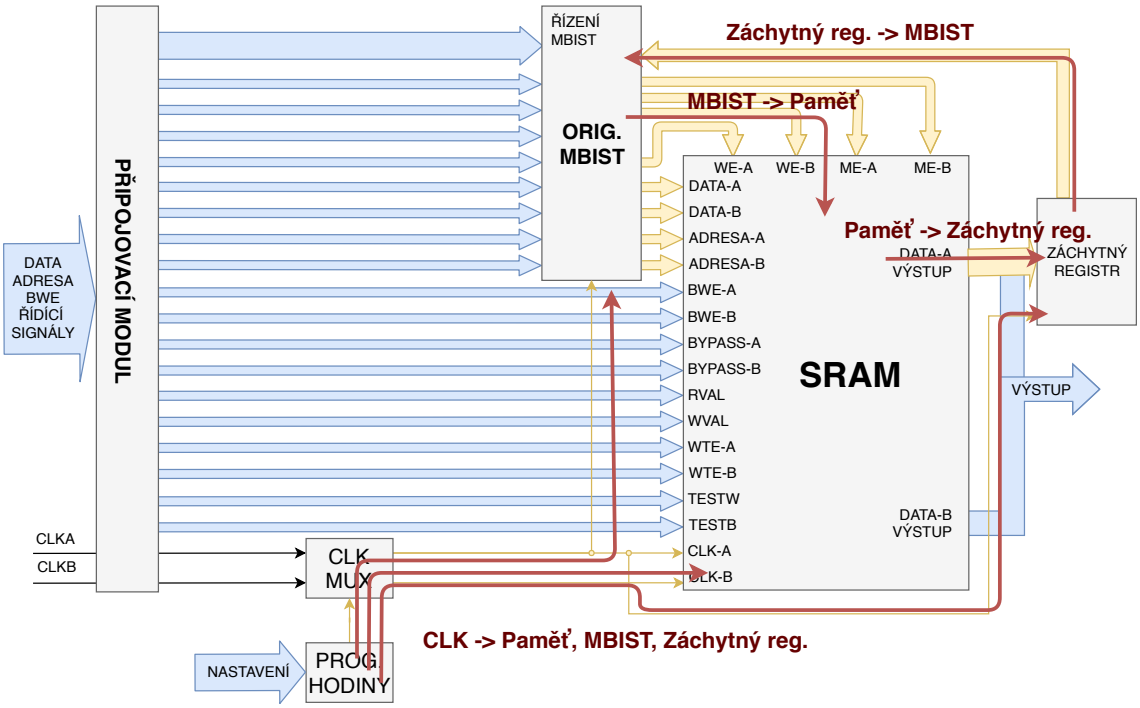
Obr. 8.3: Kritické časové cesty pro modul SHTC

Tab. 8.4: Změny v kritických cestách modulu SHTC-SP a SHTC-DP

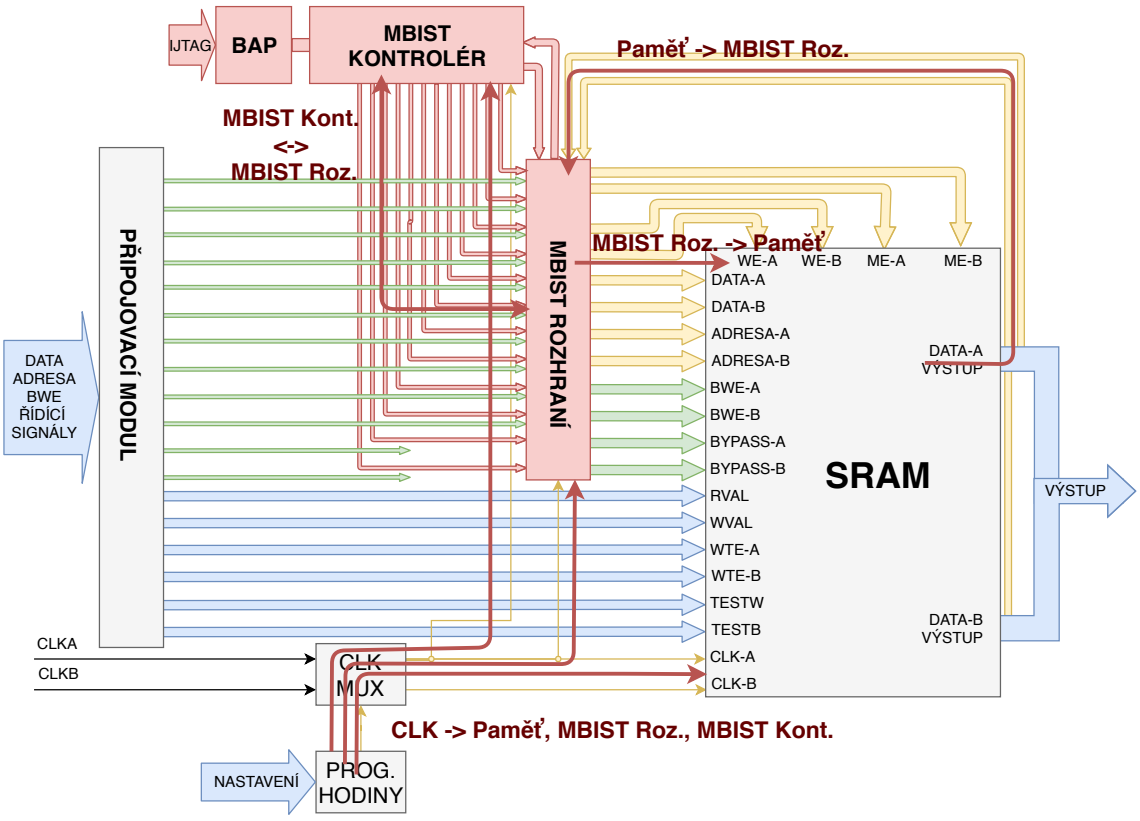
Cesta	**Změna v %**		
	CLK->Vstup-PZ	Vstup-PZ->Vstup-Paměti	ZV->ZV
Addr-0	-	0	0
Addr-1	-	0	0
Addr-2	-	0	0
Addr-3	-	0	0
Bwe-0	-	0	0
Bwe-1	-	0	0
Data-0	-	0	0
Data-1	-	0	0
Me	-	0	0
We	-	0	0
Wte	-	0	0
CLK	-	0	0

V STA pro modul MFC nebyly nalezeny chyby pro funkční testování, ale byly nalezeny pro charakterizaci. Časování kritických cest je porovnáváno s originálním MBISTem z obr. 8.4 a nahrazeným MBISTem z obr. 8.5. Jejich časování jsou uvedena v tab. 8.6 a 8.5.

Cesta, která se dostala nad kritickou mez v charakterizačním módu, se týká kontroléru MBIST generovaného nástrojem Tessent MBIST. Vygenerovaný MBIST nedokáže běžet na periodě 7.5ns zadefinované v časových požadavcích pro charakterizaci v případě že čip je napájen nejmenším možným napětím a teplota dosahuje maximální povolené hodnoty. Původní MBIST má v takovém případě nejdelší časovou cestu okolo 5ns viz. tab. D.1 a vygenerovaný MBIST s přejatým algoritmem a výchozím nastavením se pohybuje okolo 10+ ns, což je 2x větší než původní. Pro zlepšení se tedy využilo možnosti přidání (pipline) registrů na výstupu MBISTu. Dále se zvolil méně složitý SMARCH algoritmus podobný složitosti algoritmu CMARCH. Byla také nastavena syntéza na maximální úsilí. Tím bylo dosaženo zkrácení časové cesty na 8.23ns, viz tab. D.2, což je pořád nad kritickou mez. Další nastavení nástroje na zlepšení nejsou v manuálu uvedeny. Některé časové cesty mezi MBIST kontrolérem a MBIST rozhraním také překročily specifikovanou časovou mez. Takový MBIST pro charakterizaci frekvence nelze použít. Bylo by to možné, pokud testované paměti fungují na periodách větších než 9 ns.



Obr. 8.4: Kritické časové cesty pro orig. modul MFC



Obr. 8.5: Kritické časové cesty pro generovaný modul MFC

Tab. 8.5: Časy v kritických cestách modulu MFC-DP

Cesta	**Čas v ns**	
	Orig.	Náhrada
Paměť-> Záchytný reg./MBIST Roz.	6.55	7.51
Orig.MBIST/MBIST kontrolér	5.06	8.23
Orig.MBIST/MBIST rozhraní -> paměť	2.52	3.41
Záchytný reg.-> orig.MBIST	4.32	X
CLK -> Paměť, MBIST, Záchytný reg.	-	X
MBIST Kont. <-> MBIST Roz.	X	7.80
CLK -> Paměť, MBIST Roz., MBIST Kont.	X	-

Tab. 8.6: Časy v kritických cestách modulu MFC-SP

Cesta	**Čas v ns**	
	Orig.	Náhrada
Paměť-> Záchytný reg./MBIST Roz.	7.81	8.23
Orig.MBIST/MBIST kontrolér	5.32	8.74
Orig.MBIST/MBIST rozhraní -> paměť	3.13	3.81
Záchytný reg.-> orig.MBIST	4.32	X
CLK -> Paměť, MBIST, Záchytný reg.	-	X
MBIST Kont. <-> MBIST Roz.	X	8.26
CLK -> Paměť, MBIST Roz., MBIST Kont.	X	-

V STA pro modul FUNC-MEM nebyly nalezeny chyby.

8.4 Rychlost funkčního testování

Čip byl navržen tak, aby mohl běžet na periodě 110ns, ale jeho původní funkční testy jsou prováděny s periodou 2us, takto velká rezerva má zajistit správnou propagaci signálů z externího testeru. Jelikož byla do čipu vložena testovací logika, která všechny externí signály generuje interně kromě hodinového signálu, je možně, že testovací perioda může být nižší, což by znamenalo urychlení funkčních testů. To ale nebylo vyzkoušeno a proto se bude počítat s periodou 2us. Vykonávací doba algoritmu je stejná jak na externím testeru, tak na interním BISTu, pokud používáme stejné hodinové periody. Před jeho vykonáním se ale musí čip nastavit. V tom má MBIST nevýhodu, protože nastavení probíhá sériově přes JTAG v několika krocích,

a to se 4x pomalejší periodou než je testovací, po skončení testu se musí výsledky vysunout. Externí tester pouze nastaví pár registrů předtím než začne testovací algoritmus a také nemusí výsledky vysunovat, jelikož je má ihned na výstupu. Nastavení MBISTu tak může zabírat okolo 10% času jeho algoritmu. Toto procento je závislé na velkosti paměti, čím větší paměť, tím menší vliv má nastavení na celkovou dobu vykonání testu. V tomto aspektu má externí tester výhodu. Výhodou MBISTu je testování pamětí paralelně namísto původního sériového testování. Pokud budou testovány všechny paměti zaráz, tak paměť s nejvíce slovy bude udávat celkový čas pro testování, jelikož ostatní menší paměti budou již dokončené. Pokud by mělo všech 14 pamětí stejnou velikosti, tak by paralelizace přinesla až teoretické 14násobné zrychlení oproti původnímu testování, ale protože paměti nejsou velikostně stejné, přináší okolo 5násobného (8192*2+4096*2+2560*4+2048*4+32+16)/8192 zrychlení. Nastává otázka, jestli napájení čipu dokáže podporovat operace všech pamětí zaráz. Pokud by šly testovat zaráz pouze dvě paměti na tomto čipu, bylo by možné dosáhnout 2násobného zrychlení. Napájení již ale tato práce neřešila.

9 ZÁSADY NÁVRHU PRO POUŽITÍ TESSENT MBIST

První zásada se týká pamětí obsahujících speciální porty, které musí být ve výchozím stavu během používání paměti. Toto nastavení musí proběhnout přes JTAG. První řešení, nastavit permanentně výchozí hodnotu portu ve zdrojovém RTL kódu. Druhé řešení, vytvořit multiplexory ve zdrojovém RTL kódu s výchozí hodnotou portu a jejich společný řídící signál vyvést na port modulu a port nastavit na průchozí hodnotu, dokud nedojde k vložení testovací logiky, při kterém tento signál nástroj jednoduše zachytí, protože při syntéze nebyl vstupní port optimalizován. Třetí řešení, pozměnit šablony paměti pro port na funkci user1 a nastavit výchozí hodnoty. Porty s více bity jde nastavit do samých nul nebo do samých jedniček, pokud je požadována jiná hodnota, musí se použít druhé řešení nebo rozdělit vícebitový port na jednobitové porty pomocí obálky, více o této limitaci se nachází v kapitole 7.1.

Druhá zásada se týká výstupu paměti. Pokud se do cesty výstupu nemá vložit žádné hradlo, musí se vypnout nastavení scan_bypass_logic.

Třetí zásada se týká vstupu paměti. Nástroj vždy vloží MBIST rozhraní přímo před paměť, což může způsobit nechtěná hradla a zpoždění v cestě. První řešení, po vložení testovací logiky spustit přepojovací skript. Druhé řešení, paměť a logiku, která má být na vstupu nebo i na výstupu zapouzdřit, do jednoho RTL modulu a tento modul vydávat za paměť. Modul musí mít stejnou bitovou šířku portů jak paměť uvnitř a porty jako DATA-A a DATA-B nesmí být spojeny do jednoho. Modul vydávaný za paměť musí mít vždy jednu paměť. Šablona modulu se bude vytvářet na základě šablony paměti, porty, kterých se nástroj nesmí dotknout, se označí jako none. Modul bude mít jeden port s funkcí user, pokud se modul potýká s první zásadou, a pokud logika, která byla ochráněna před špatným vložením, musí být nastavena tak, aby cesta z modulu do paměti byla průchozí.

Pro ATC charakterizaci se použila první a druhá zásada a nebyly nutné skoro žádné zásady do zdrojových RTL souborů. Pro SHTC charakterizaci se použily všechny zásady a byly provedeny zásahy do zdrojových RTL souborů.

10 ZÁVĚR

Bylo navrženo řešení testování pamětí na reálném návrhu čipu za použíti nástroje Tessent MemoryBIST. Implementovat MBIST se podařilo na všechny SRAM paměti. Testovací logika byla úspěšně verifikována logickou simulací a statickou analýzou. MBIST nyní dokáže otestovat čip 2x až 5x rychleji než původně. Násobek je dán možností testovat několik pamětí zaráz, které ale závisí na možnostech napájení. V budoucnu se může také ověřit použití kratší periody do MBISTu pro zrychleni testů. Pro budoucí použití nástroje na novém návrhu, byly vytvořeny zásady pro vkládání testovací logiky pro charakterizační moduly, které používají paměti nezvyklým způsobem.

Nástroj není ale zcela bez problémů. Bylo nutné se vypořádat s limitacemi samotného nástroje, kdy se muselo uchýlit k vytváření obálek modulů pro přizpůsobení jeho limitacím. Nebo bylo zapotřebí napsat skript, který přepojil testovací modulu na jiné místo. Testovací logika nezpůsobuje velký nárůst plochy pro větší paměti, kdy poměr Paměť/MBIST je velký ale, při malých pamětích může. Při STA analýze byl pozorován předpokládaný nárůst časových cest z a dovnitř pamětí. Při vytváření MBISTu s maximální teoretickou frekvencí paměti způsobila jeho komplexnost nemožnost zkrátit jeho nejdelší časovou cestu pod 8.23 ns. Což ale MBIST znemožňuje použít pro účel charakterizace maximální frekvence pro paměti v čipu.

Nástroj dokáže automaticky vytvářet testovací vektory pro použití v ATE. Jediná potíž je v tom, že podporuje pouze vytváření statických testů. Pro účely diagnostiky či charakterizace je potřeba, aby testy reagovaly na výstupní data během testu. Takový charakterizační program by šel napsat ze souborů(PDL ICL) vygenerovaným nástrojem a následně převést za pomocí interpretoru na použití v ATE. Ale aktuálně takový interpretor není ve firmě dostupný a varianta zdarma nebyla nalezena. Způsob upravení statických testovacích vektorů pro diagnostiku je možný, ne ale zcela jednoduchý, protože testovací vektory jsou určeny pro komunikační rozhraní JTAG a jeho síť konfigurovatelných cest. Nástroj podporuje interaktivní diagnostiku s hardwarovým komunikačním zařízením ale, pro účely charakterizace stovky čipů je takový způsob nevhodný.

Nástroj Tessent MBIST(verze 2017) se tak hodí pro rychlé funkční testování pamětí za pomoci statických vektorů. Jeho výhody spočívají v automatickém generování MBIST logiky, které se můžou rychle přizpůsobovat měnícímu se návrhu.

LITERATURA

[1] Adams, R. D.: *High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test*, *Frontiers in Electronic Testing*, ročník 22A. Springer US, první vydání, 2003, 250 s.

[2] Alberto Bosio, P. G. S. P. A. V., Luigi Dilillo: *Advanced Test Methods for SRAMs*. Springer US: Springer, první vydání, 2010, ISBN 978-1-4419-0937-4, 171 s.

[3] Bostock, M.: Why Use Make. 2013.
URL `https://web.archive.org/web/20180119195106/https://bost.ocks.org/mike/make/`

[4] Cypress: *CY6264 8K × 8 Static RAM*. Říjen 2016, rev. *G.
URL `http://www.cypress.com/file/43026/download`

[5] Hamdioui, S.; van de Goor, A. J.; Rodgers, M.: March SS: a test for all static simple RAM faults. In *Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing (MTDT2002)*, 2002, ISSN 1087-4852, s. 95–100, doi:10.1109/MTDT.2002.1029769.

[6] IJTAG: IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device. *IEEE Std 1687-2014*, Dec 2014: s. 1–283, doi:10.1109/IEEESTD.2014.6974961.

[7] JTAG: IEEE Standard for Test Access Port and Boundary-Scan Architecture. *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, May 2013: s. 1–444, doi:10.1109/IEEESTD.2013.6515989.

[8] Mentor Graphics Corporation: *Tessent MemoryBIST User's Manual*. Červen 2017, rev. 5.

[9] Mentor Graphics Corporation: *Tessent Shell Reference Manuall*. Červen 2017, rev. 5.

[10] Nadeau-Dostie, B.: *Design for AT-Speed Test, Diagnosis and Measurement*. Springer US, 2002, ISBN 978-0-7923-8669-8, 239 s.

[11] Wikipedia contributors: Built-in self-test — Wikipedia, The Free Encyclopedia. 2018, [Online; accessed 30-April-2018].
URL `https://en.wikipedia.org/w/index.php?title=Built-in_self-test&oldid=819260418`

SEZNAM PŘÍLOH

A Algoritmy **72**

B Tessent MBIST **74**

C Implementace DFT pro SP paměti **76**

D MBIST časové cesty **83**

A ALGORITMY

Tab. A.1: Jednotlivé algoritmy

#	**Test**	**Algoritmus**	**Zdroj**
1	SCAN	$\{\Uparrow (W0); \Uparrow (R0); \Uparrow (W1); \Uparrow (R0);\}$	[2]
2	MATS +	$\{\Updownarrow (W0); \Uparrow (R0, W1); \Downarrow (R1, W0);\}$	[2]
3	MATS++	$\{\Updownarrow (W0); \Uparrow (R0, W1); \Downarrow (R1, W0, R0);\}$	[2]
4	March C-	$\{\Updownarrow (W0); \Uparrow (R0, W1); \Downarrow (R1, W0); \Downarrow (R0, W1);$ $\Downarrow (R1, W0); \Updownarrow (R0);\}$	[2]
5	PMOVI	$\{\Downarrow (W0); \Uparrow (R0, W1, R1); \Uparrow (R1, W0, R0);$ $\Downarrow (R0, W1, R1); \Downarrow (R1, W0, R0);\}$	[2]
6	March G	$\{\Updownarrow (W0); \Uparrow (R0, W1, R1, W0, R0, W1); \Uparrow (R1, W0, W1);$ $\Downarrow (R1, W0, W1, W0); \Downarrow (R0, W1, W0); \Uparrow (R0, W1, R1);$ $\Uparrow (R1, W0, R0);\}$	[2]
7	Butterfly	$\{\Uparrow (W0); \Uparrow (W1b, o(R0), R1b, W0b);$ $\Uparrow (W1); \Uparrow (W0b, o(R1), R0b, W1b;\ \}$	[2]
8	Galpat	$\{\Uparrow (W0); \Uparrow b(W1b, \Uparrow -b(R0, R1b); W0b;)$ $\Uparrow (W1); \Uparrow (W0b, \Uparrow -b(R0, R1b);, W1b);\ \}$	[2]
9	Walking	$\{\Uparrow (W0); \Uparrow b(W1b, \Uparrow b(R0); R1b; W0b;)$ $\Uparrow (W1); \Uparrow (W0b, \Uparrow b(R0); R1b, W1b);\}$	[2]
10	March B	$\{\Updownarrow (W0); \Uparrow (R0, W1, R1, W0, R0, W1);$ $\Uparrow (R1, W0, W1); \Downarrow (R1, W0, W1, W0);$ $\Downarrow (R0, W1, W0);\}$	[5]
11	March U	$\{\Updownarrow (W0); \Uparrow (R0, W1, R1, R0); \Uparrow (R0, W1);$ $\Downarrow (R1, W0, R0, W1); \Downarrow (R1, W0);\}$	[5]
12	March LR	$\{\Updownarrow (W0); \Downarrow (R0, W1); \Uparrow (R1, W0, R0, W1);\ \Uparrow (R1, W0);$ $\Uparrow (R0, W1, R1, W0); \Uparrow (R0);\}$	[5]
13	March SR	$\{\Updownarrow (W0); \Uparrow (R0, W1, R1, W0); \Uparrow (R0, R0); \Uparrow (W1);$ $\Downarrow (R1, W0, R0, W1); \Downarrow (R1, R1);\}$	[5]
14	March SS	$\{\Updownarrow (W0); \Uparrow (R0, R0, W0, R0, W1);$ $\Uparrow (R1, R1, W1, R1, W0); \Downarrow (R0, R0, W0, R0, W1);$ $\Downarrow (R1, R1, W1, R1, W0); \Updownarrow (R0);\}$	[5]

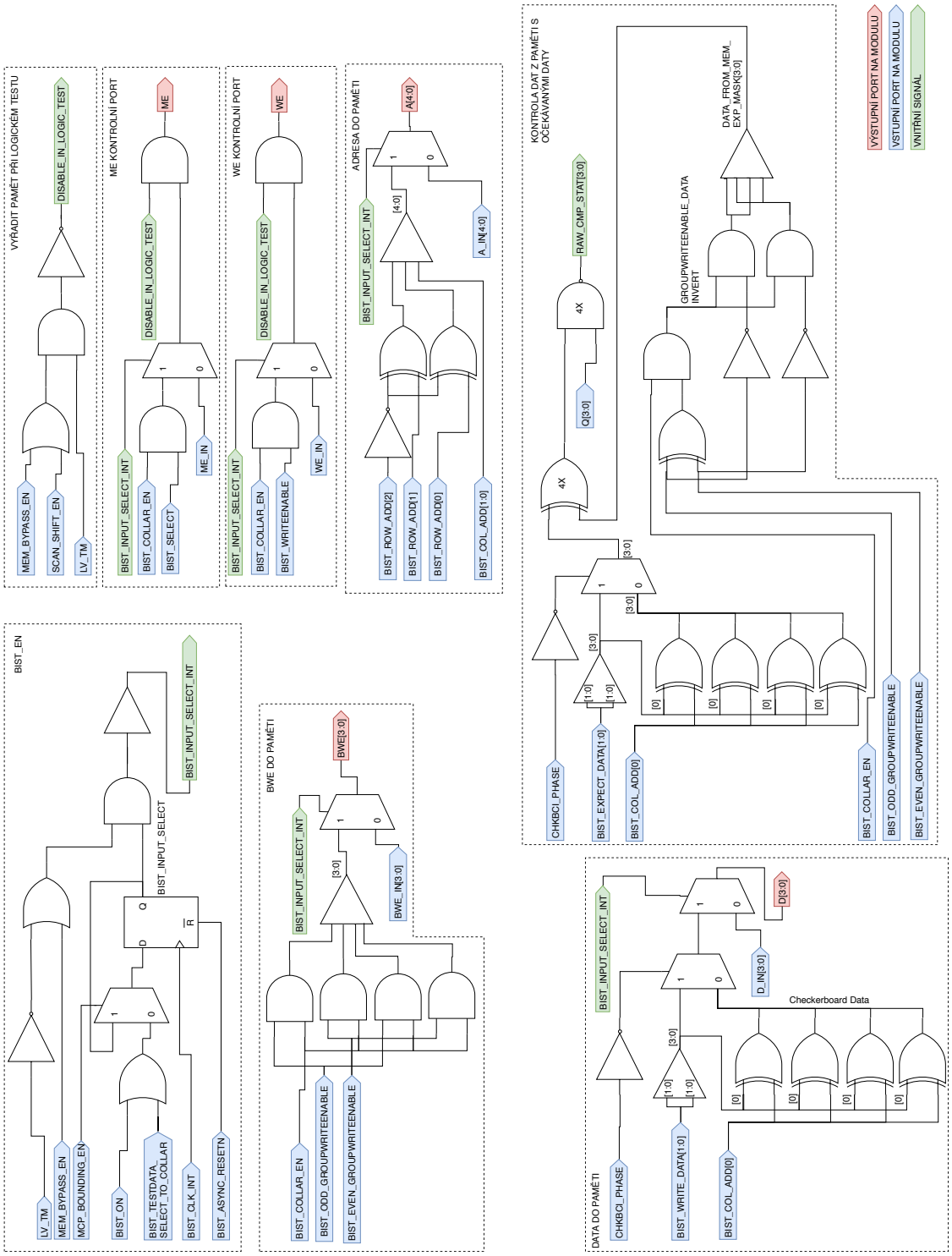
Tab. A.2: Pokrytí chyb pro jednu buňku

#	Test	Délka	SF	TF	WDF	RDF	DRDF	IRF	FC
1	SCAN	4n	2/2	1/2	0/2	2/2	0/2	2/2	7/12
2	MATS +	5n	2/2	1/2	0/2	2/2	0/2	2/2	7/12
3	MATS++	6n	2/2	2/2	0/2	2/2	0/2	2/2	8/12
4	March C-	10n	2/2	2/2	0/2	2/2	0/2	2/2	8/12
5	PMOVI	13n	2/2	2/2	0/2	2/2	2/2	2/2	10/12
6	March G	23n	2/2	2/2	0/2	2/2	1/2	2/2	9/12
7	Butterfly	16n	2/2	2/2	0/2	2/2	0/2	2/2	8/12
8	Galpat	6n+4nRC	2/2	2/2	0/2	2/2	2/2	2/2	10/12
9	Walking	8n+2nRC	2/2	2/2	0/2	2/2	0/2	2/2	8/12
10	March B	17n	2/2	2/2	0/2	2/2	0/2	2/2	8/12
11	March U	13n	2/2	2/2	0/2	2/2	0/2	2/2	8/12
12	March LR	14n	2/2	2/2	0/2	2/2	0/2	2/2	8/12
13	March SR	14n	2/2	2/2	0/2	2/2	2/2	2/2	10/12
14	March SS	22n	2/2	2/2	2/2	2/2	2/2	2/2	12/12

Tab. A.3: Pokrytí chyb pro dvě buňky

Test	CFst	CFds			CFtr	CFwd	CFrd	CFdr	CFir	FC
		xrx	xr-x	xwx						
SCAN	5/8	2/8	1/8	0/8	2/8	0/8	4/8	0/8	4/8	18/72
MATS+	6/8	3/8	3/8	0/8	2/8	0/8	4/8	0/8	4/8	22/72
MATS++	6/8	3/8	3/8	0/8	4/8	0/8	4/8	0/8	4/8	24/72
March C-	8/8	8/8	8/8	0/8	8/8	0/8	8/8	0/8	8/8	48/72
PMOVI	8/8	8/8	7/8	0/8	8/8	0/8	8/8	6/8	8/8	53/72
March G	8/8	7/8	8/8	0/8	7/8	0/8	8/8	2/8	8/8	48/72
Butterfly	8/8	4/8	4/8	0/8	4/8	0/8	8/8	0/8	8/8	36/72
Galpat	8/8	8/8	4/8	0/8	4/8	0/8	8/8	4/8	8/8	44/72
Walking	8/8	8/8	4/8	0/8	4/8	0/8	8/8	0/8	8/8	40/72
March B	6/8	7/8	8/8	0/8	4/8	0/8	4/8	0/8	4/8	33/72
March U	8/8	8/8	8/8	0/8	8/8	0/8	8/8	0/8	8/8	48/72
March LR	8/8	8/8	8/8	0/8	8/8	0/8	8/8	0/8	8/8	48/72
March SR	8/8	8/8	8/8	0/8	8/8	0/8	8/8	6/8	8/8	54/72
March SS	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	72/72

B TESSENT MBIST

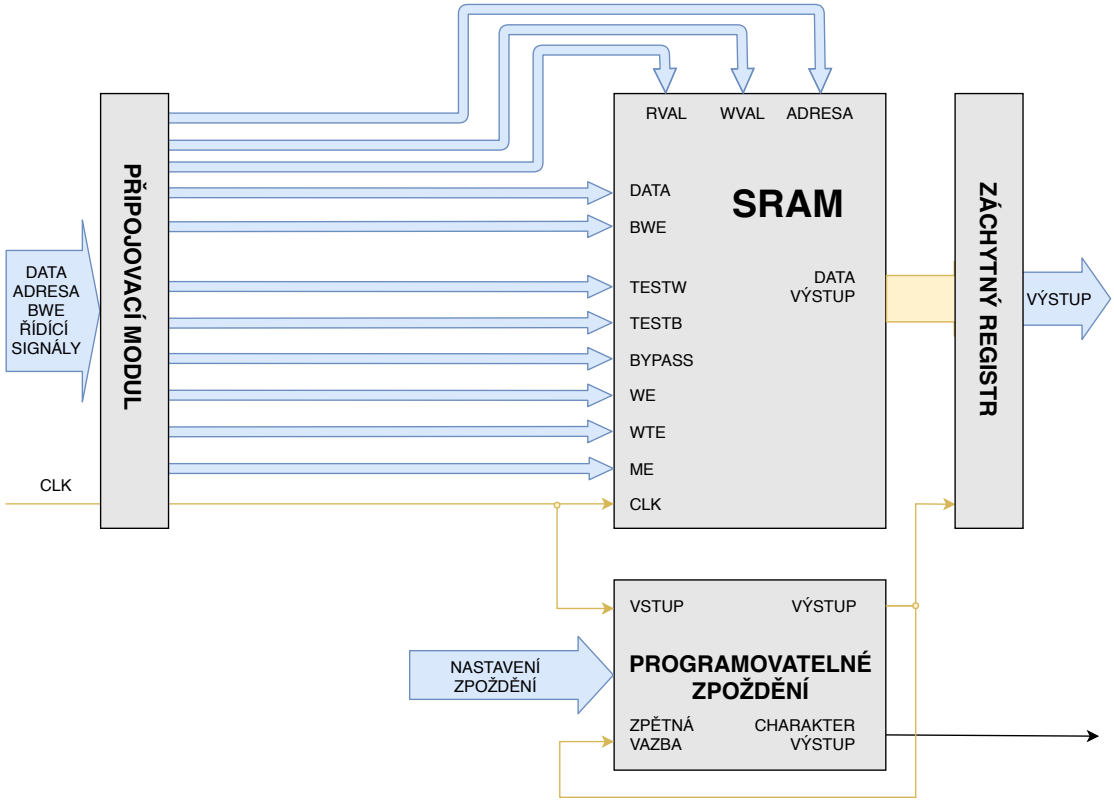


Obr. B.1: Ukázka MBIST rozhraní pro paměť s 4-bitovou sběrnicí

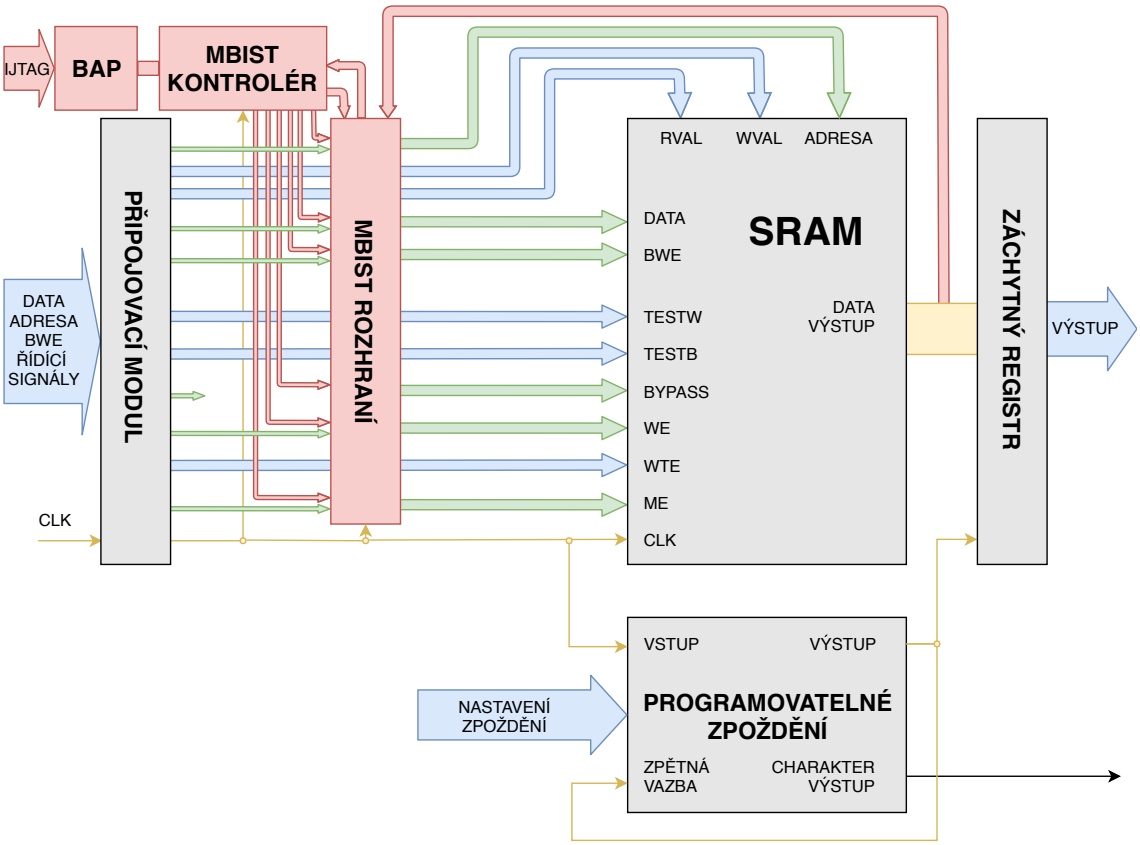
```
Core(SP SRAM s 2560 Slovy po 48 bitech)
└── Memory
    ├── Algorithm : SMarchCHKBci;
    ├── BitGrouping : 1;
    ├── DataOutHoldWithInactiveReadEnable : on;
    ├── DataOutStage : none;
    ├── LogicalPorts : 1RW;
    ├── MemoryHoldWithInactiveSelect : on;
    ├── MemoryType : sram;
    ├── MilliWattsPerMegaHertz : auto;
    ├── NumberOfBits : 48;
    ├── NumberOfWords : 2560;
    ├── OperationSet : Sync;
    ├── TransparentMode : syncmux;
    ├── Port(CLK)
    │   ├── Direction : input;
    │   ├── Function : Clock;
    │   └── Polarity: activeHigh;
    ├── Port(A)
    │   ├── BusRange : [13:0];
    │   ├── Direction : input;
    │   ├── Function : Address;
    │   └── Polarity: activeHigh;
    ├── Port(D)
    │   ├── BusRange : [47:0];
    │   ├── Direction : input;
    │   ├── Function : Data;
    │   └── Polarity: activeHigh;
    └── Port(Q)
        ├── BusRange : [47:0];
        ├── Direction : output;
        └── Function : Data;
```

Obr. B.2: Šablona paměti pro MBIST

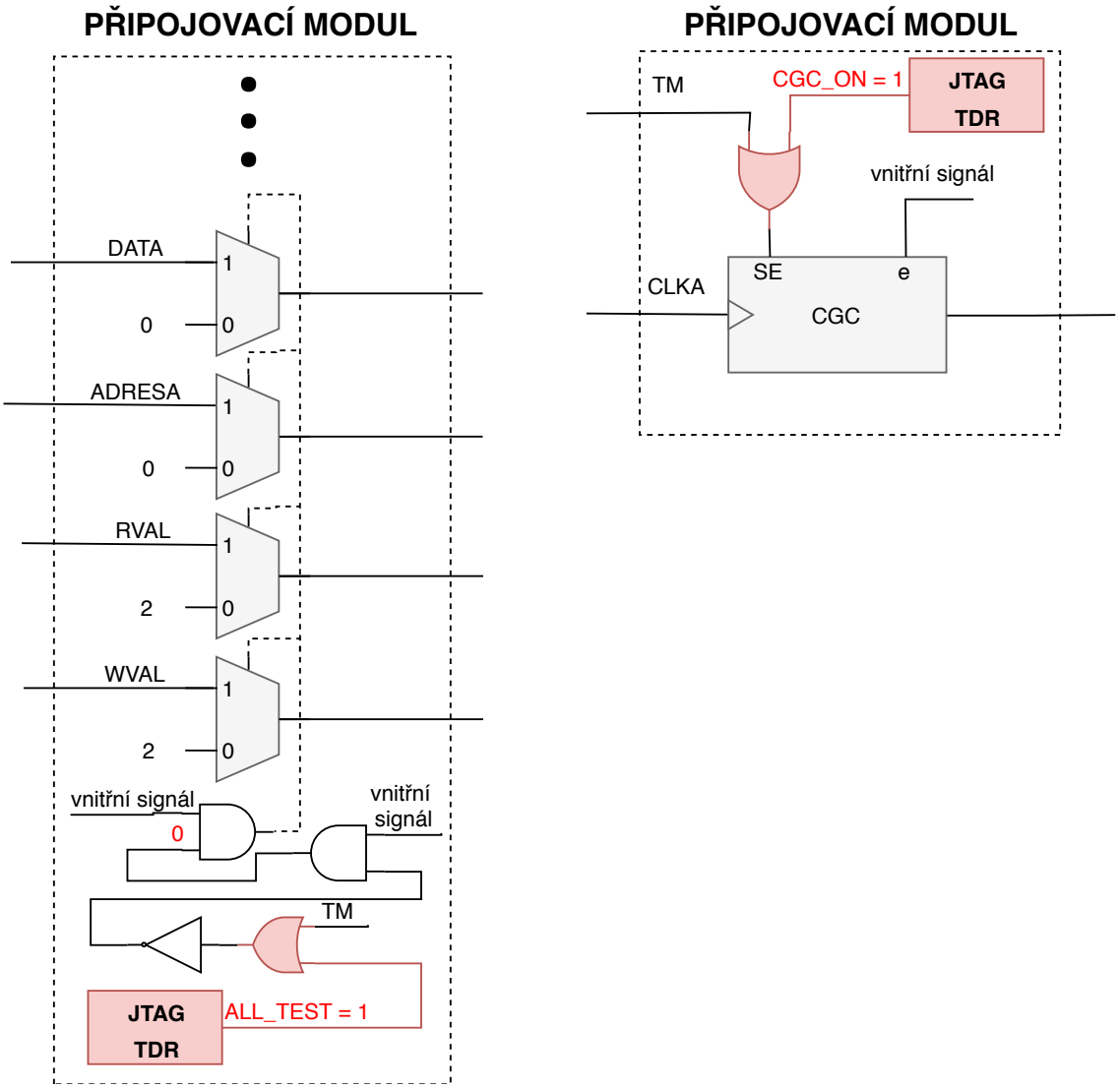
C IMPLEMENTACE DFT PRO SP PAMĚTI



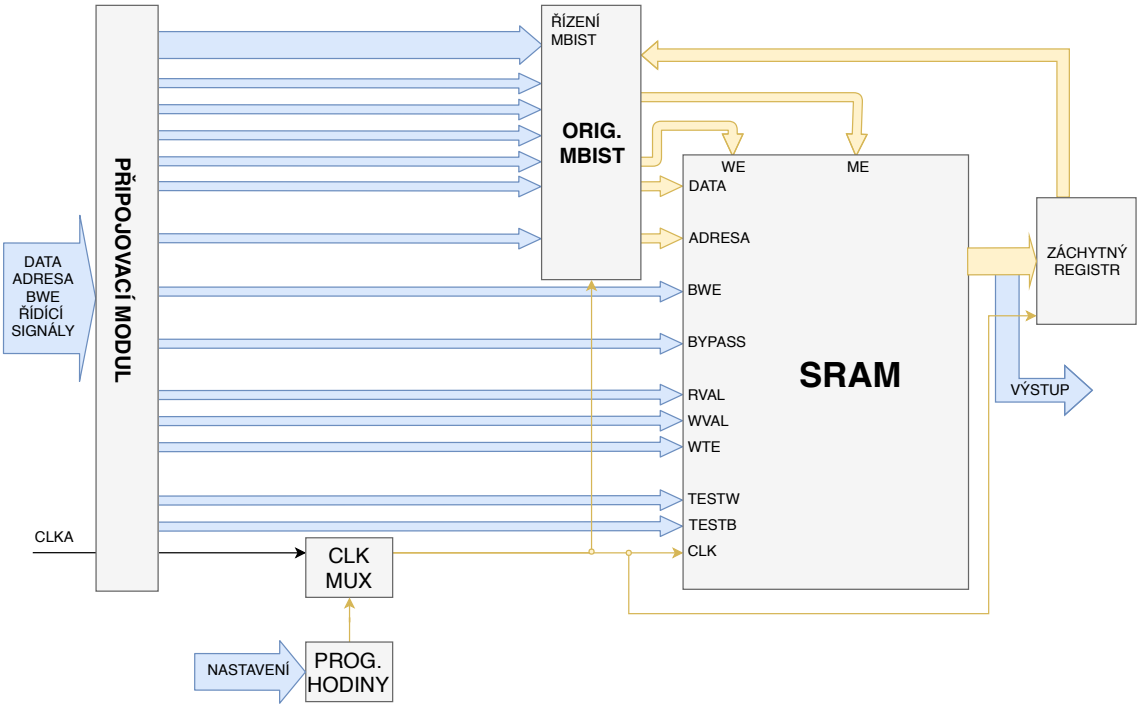
Obr. C.1: ATC-SP blokové schéma pro charakterizaci přístupového času jednoportové paměti



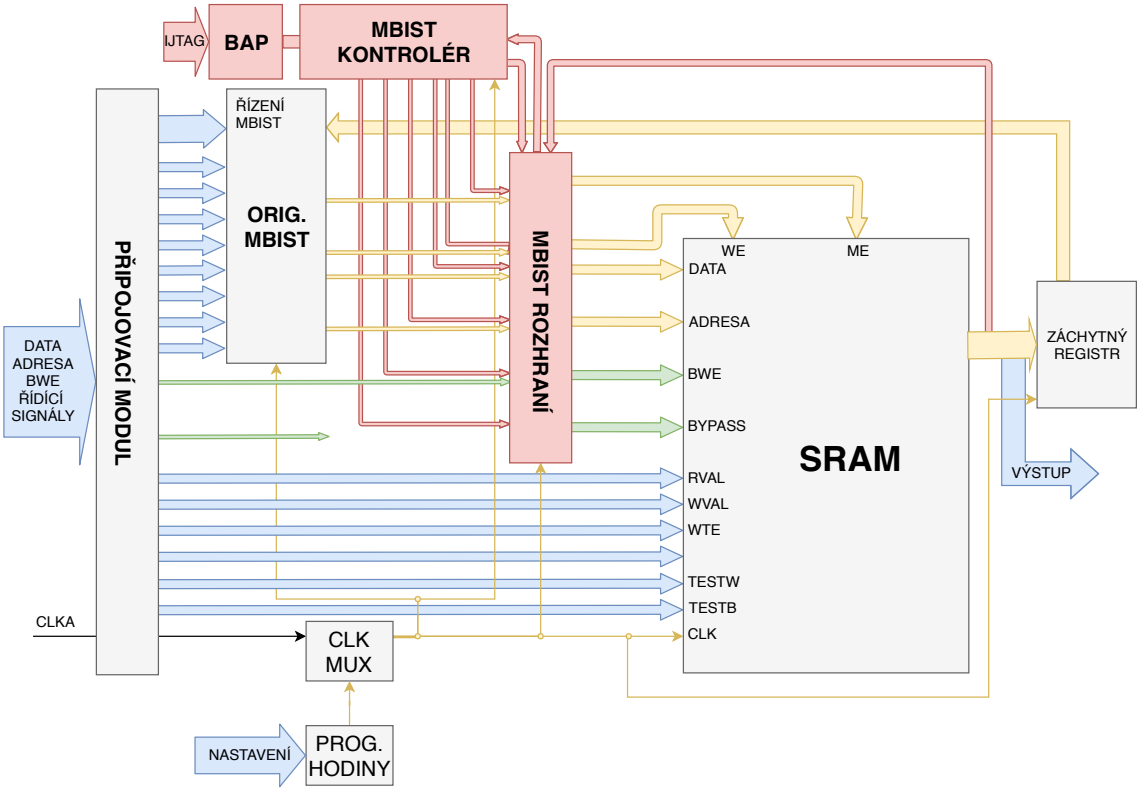
Obr. C.2: ATC-SP blokové schéma po vložení testovací logiky paměti



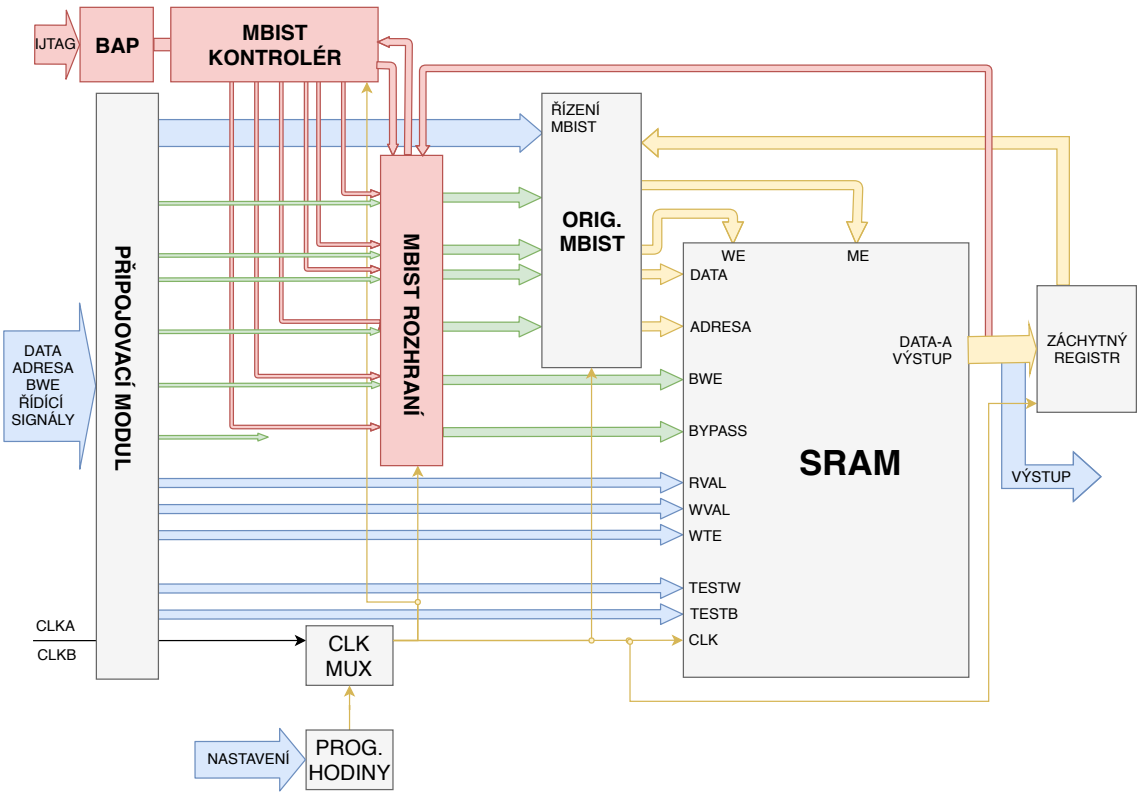
Obr. C.3: ATC-SP připojovací modul



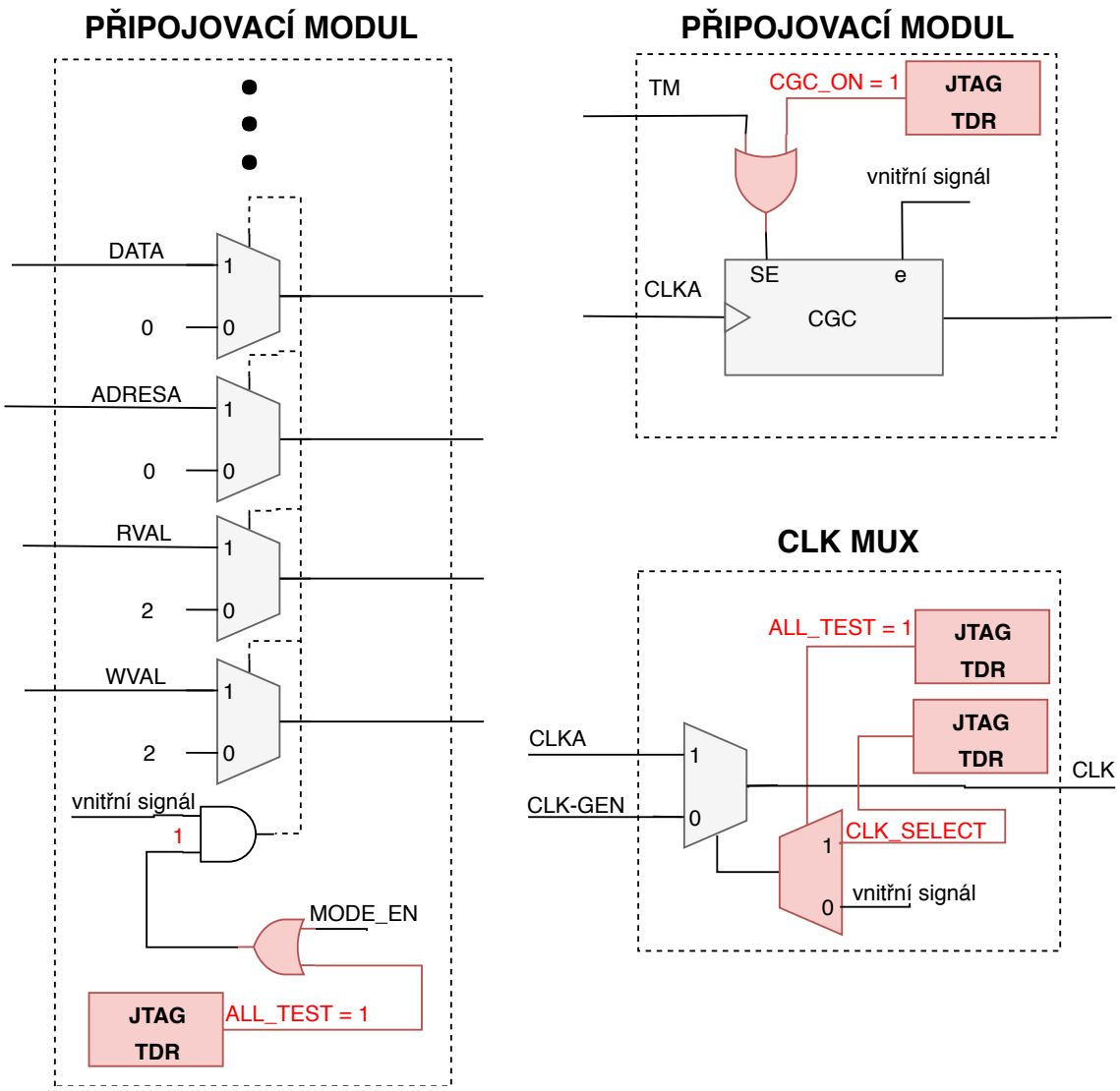
Obr. C.4: MFC-SP blokové schéma



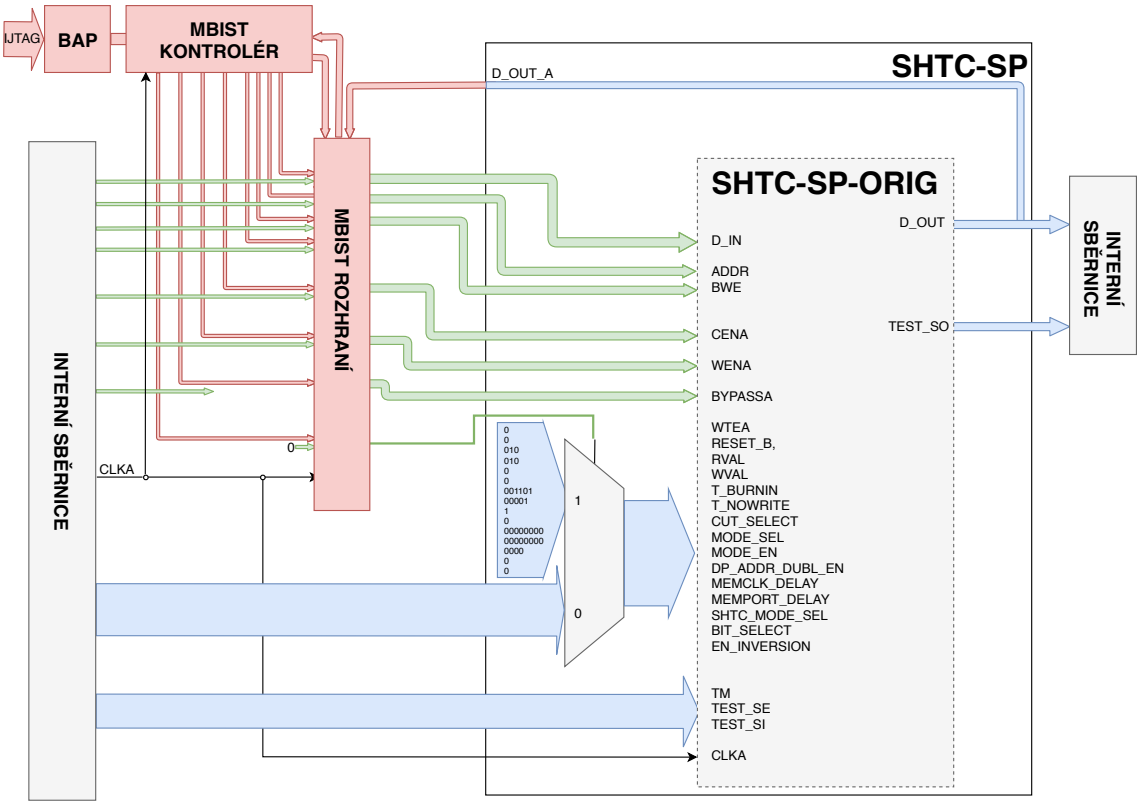
Obr. C.5: MFC-SP blokové schéma po vložení testovací logiky paměti



Obr. C.6: MFC-SP požadovaná testovací logika paměti



Obr. C.7: MFC-SP připojovací modul



Obr. C.8: SHTC-SP blokové schéma po vložení testovací logiky paměti

D MBIST ČASOVÉ CESTY

Point	Trans	Incr	Path
clock INT_CLK (rise edge)	0.350	0.000	0.000
clock network delay (ideal)		0.000	0.000
inst_ram_bist/inst_addr_fsm_ADDR_reg_2/CLK (sdffrp_4_1)	0.350	0.000	0.000 r
inst_ram_bist/inst_addr_fsm_ADDR_reg_2/QB (sdffrp_4_1)	0.292	1.322	1.322 r
inst_ram_bist/g12425/X (inv_3_1)	0.106	0.196	1.517 f
inst_ram_bist/g12781/X (and2_4_1)	0.134	0.485	2.003 f
inst_ram_bist/g12034/X (nand2_2_1)	0.408	0.363	2.366 r
inst_ram_bist/g11916/X (nor2_4_1)	0.215	0.354	2.721 f
inst_ram_bist/g11818/X (nand2_4_1)	0.282	0.331	3.052 r
inst_ram_bist/g12786/X (nor2i_4_1)	0.185	0.296	3.348 f
inst_ram_bist/g11487/X (and2_4_1)	0.193	0.603	3.952 f
inst_ram_bist/g11247/X (nand2_4_1)	0.231	0.285	4.236 r
inst_ram_bist/g11227/X (aoi21c_4_1)	0.179	0.256	4.492 f
inst_ram_bist/g11212/X (aoai211_2_1)	0.453	0.572	5.064 r
inst_ram_bist/inst_addr_fsm_ADDR_reg_11/D (sdffrp_4_1)	0.453	0.000	5.064 r
data arrival time			5.064
clock INT_CLK (rise edge)	0.050	7.500	7.500
clock network delay (ideal)		0.000	7.500
clock reconvergence pessimism		0.000	7.500
clock uncertainty		−0.010	7.490
inst_ram_bist/inst_addr_fsm_ADDR_reg_11/CLK (sdffrp_4_1)			7.490 r
library setup time		−0.887	6.603
data required time			6.603
data required time			6.603
data arrival time			−5.064
slack (MET)			1.538

Obr. D.1: Nejdelší časová cesta pro původní MBIST

Point	Incr	Path
clock INT_CLK (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
mfc_sp_tessent_mbist_controller/MBISTPG_ASYNC_INTERF/tessent_persistent_cell_MBIST_NTC_RETIMING_CELL_TCK/q_reg/CLK (dffprq_1_l)	0.00	0.00 r
mfc_sp_tessent_mbist_controller/MBISTPG_ASYNC_INTERF/tessent_persistent_cell_MBIST_NTC_RETIMING_CELL_TCK/q_reg/Q (dffprq_1_l)	1.02	1.02 f
mfc_sp_tessent_mbist_controller/MBISTPG_ASYNC_INTERF/U10/X (inv_2_l)	0.34	1.36 r
mfc_sp_tessent_mbist_controller/MBISTPG_ASYNC_INTERF/U12/X (nor2_2_l)	0.20	1.56 f
mfc_sp_tessent_mbist_controller/MBISTPG_ASYNC_INTERF/U5/X (or2_4_l)	0.50	2.06 f
mfc_sp_tessent_mbist_controller/MBISTPG_ASYNC_INTERF/tessent_persistent_cell_AND_BIST_SHIFT_SYNC/X (and2_4_l)	0.55	2.61 f
mfc_sp_tessent_mbist_controller/MBISTPG_ASYNC_INTERF/tessent_persistent_cell_AND_BIST_SETUP_SYNC_WITH_SHIFT1/X (and2_2_l)	0.52	3.13 f
mfc_sp_tessent_mbist_controller/MBISTPG_ASYNC_INTERF/U9/X (nor2_2_l)	0.26	3.39 r
mfc_sp_tessent_mbist_controller/U9/X (inv_3_l)	0.19	3.58 f
mfc_sp_tessent_mbist_controller/U7/X (inv_1_l)	0.64	4.22 r
mfc_sp_tessent_mbist_controller/MBISTPG_ADD_GEN/U103/X (inv_2_l)	0.45	4.67 f
mfc_sp_tessent_mbist_controller/MBISTPG_ADD_GEN/U3/X (nand2_2_l)	0.50	5.17 r
mfc_sp_tessent_mbist_controller/MBISTPG_ADD_GEN/U122/X (aoi31_4_l)	0.39	5.56 f
mfc_sp_tessent_mbist_controller/MBISTPG_ADD_GEN/U104/X (or2_4_l)	0.83	6.39 f
mfc_sp_tessent_mbist_controller/MBISTPG_ADD_GEN/U34/X (nor3_2_l)	0.61	7.00 r
mfc_sp_tessent_mbist_controller/MBISTPG_ADD_GEN/U72/X (ao22_1_l)	0.77	7.78 r
mfc_sp_tessent_mbist_controller/MBISTPG_ADD_GEN/U126/X (ao211_2_l)	0.45	8.23 r
mfc_sp_tessent_mbist_controller/MBISTPG_ADD_GEN/AX_ADD_REG_reg[3]/D (dffpr_1_l)	0.00	8.23 r
data arrival time	8.23	
clock INT_CLK (rise edge)	7.50	7.50
clock network delay (ideal)	0.00	7.50
clock reconvergence pessimism	0.00	7.50
clock uncertainty	−0.01	7.49
mfc_sp_tessent_mbist_controller/MBISTPG_ADD_GEN/AX_ADD_REG_reg[3]/CLK (dffpr_1_l)		7.49 r
library setup time	−0.62	6.87
data required time		6.87
data required time		6.87
data arrival time		−8.23
slack (VIOLATED)		

Obr. D.2: Nejdelší časová cesta pro generovaný MBIST